A GRAPHICAL ENVIRONMENT FOR THE SIMULATION OF PETRI NETS

by

TAN, JOO TONG

B.S., University of New Mexico, 1986

A THESIS

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1988

Approved by:

Virg Wallentine

Major Professor

Table of Contents

List of Figures

Acknowledgements

Thanks are due to Dr. Maria Zamfir for her assistance and guidance throughout the course of this thesis. I am also grateful to Dr. William Hankley and Dr. Austin Melton for serving on my advisory committee. Special thanks are due to my good friend, Saiid Paryavi, for his helpful criticism of this work. Last, but certainly not least, I like to thank my wife, Hsu Ling-Ling, for her encouragement and patience during the entire implementation and writing of this thesis, without whom I would never have made it.

CHAPTER 1

INTRODUCTION

1.1 Introduction

A Petri net is an abstract formal model of information flow. Petri nets were designed and are used mainly for modeling. In particular, Petri nets are suitable for modeling systems with independent components, especially those that may exhibit asynchronous and concurrent activities. Petri nets represent a long and sustained effort to develop concepts, theories, and tools to aid in the design and analysis of concurrent systems. Various kinds of 'real' systems can be described, analyzed, and synthesized by Petri nets. Petri nets are used in areas such as software engineering, office automation, specification and verification of protocols, databases, legal systems, and industrial as well as social systems. Using Petri nets to model systems gives the user several advantages:

1) the representation scheme provides a graphical interface that is easy to understand,

2) systems can be designed systematically because Petri nets can be synthesized in either a top-down or bottom-up approach,

3) the development of an appropriate theory allows analysis of the modeled system giving important properties regarding its behavior.

However the practical use of Petri nets relies heavily on the existence of adequate computer tools. Therefore, to facilitate the description and modeling of systems, computer tools should be available to aid the user in the construction and simulation of Petri nets. A rapidly growing need for computer assistance persists in the drawing, simulation, and analysis of the various types of Petri nets.

The purpose of our work is to design and implement a Petri net tool package for modeling, simulation, and analysis of concurrent systems. In Sections 1.3 and 1.4 we discuss in detail the Petri net tool package and its implementation using the object-oriented

style of programming. In the first section of chapter 2, we look briefly into the history and development of Petri nets. Basic Petri net concepts that are used in this thesis will be formally defined in Section 2.2. We developed a Petri net tool package called PETRISYS which consists of a net editor, a simulator, an analyzer, and a syntax-checker. We describe PETRISYS in chapter 3 and conclude that chapter with an example. In Chapter 4 we present the implementation issues of our system. Conclusions, possible extensions to this system, and the value of our work can be found in Chapter 5. A bibliography follows this chapter. Finally, an appendix containing a sample of the implementation can be found at the end of this thesis.

1.2 Computer tools for the creation, modification, and analysis of Petri nets

Computer systems are commonly viewed in the computing environment as an information processing system or as a communication system. For our work we view the computer system as a set of tools. By taking the tool perspective when designing a computer system for Petri nets, programs can be considered as sets of tools that are under the control of a user. Computer tools may effectively assist users during the various phases of modeling. A sophisticated tool should be able to assist users in coping with the many details of a large description in a simple way. Tools may be designed to model different net types or only one net type. Petri net tools offer the user a number of different advantages:

1) the possibility to create better and faster results; a computer-based drawing system allows the user to produce drawings that are high in quality and precision in a shorter period of time,

2) the opportunity to make interactive presentations of the results; this is particularly useful if the system to be modeled is large and is almost impossible to present on a fixed medium such as sheets of paper,

3) the freedom to apply certain kinds of analysis systems without having a detailed knowledge of the underlying theory; the computer system normally takes care of the technical aspects of the underlying Petri net theory and performs the necessary calculations, while the user has only to decide what is to be done,

4) the ability to hierarchically structure the process by which he obtains the results; in this case, an entire net may be replaced by a single place or transition for modeling at a more abstract level or places and transitions may be replaced by subnets to provide more detailed modeling,

5) the capability to produce end-products of high quality; this is aided by the use of letter-quality printers and laser printers.

Certain characteristics form a basic set of requirements for a good tool. We list the most general ones here:

1) A good tool provides a "helpful" interface for the casual user and an "effective" interface for the frequent or more skillful user,

2) A novice user should be able to start doing simple tasks after a few hours of training, whereas the more experience user should be able to continue improving the way in which he masters the system. This improvement may involve a gain in speed as well as an increase in the quality of the drawing,

3) Good tools maintain a "suitable" balance between the structure that is imposed by the system and the expressive freedom that is offered to the user,

4) Good tools constitute a carefully designed balance between general applicable functions and special purpose functions.

Depending on the application, tools should have a balanced combination of editing and analysis functions to fit the specific need. Graphical workstations provide the opportunity for users to work with both the textual and graphical representations of Petri nets. Graphical workstations also facilitate the process of creating interfaces that are highly interactive; thus, easing the process of learning, using, and understanding different application programs. To effectively apply Petri nets, the following kinds of tools are now available:

1) Graphical Editors

A graphical editor allows users to draw and modify Petri nets by directly working with their graphical representation. A menu of commands or options is usually available for the user to make selections from. The use of a mouse as an input device has become common practice in interactive graphical environments. Users can see the immediate results of their work displayed on the screen. Graphical net editors provide very accurate and high-precision drawings that is close in form to the final printed product. These editors will often have a high degree of expressive power, i.e. the user usually has a number of ways to draw a net. The graphical editor should also be able to assist users in handling drawings of large Petri nets in a well-structured and hierarchical way. Two good graphical net editors are [Beau83] and [Mont83].

2) Textual Editors

Textual editors allow users to construct and modify Petri nets by working with their textual representation. This type of editor should have many of the facilities that is

provided by normal word processing systems. In addition, textual editors should be able to recognize the structure of Petri nets and check that the constructed nets are consistent with net type(s) that are supported by the system. This kind of editor is usually not as appealing as graphical editors because users are forced to remember code commands. These commands are sometimes quite long and complicated because they may involve a combination of keys.

3) Analysis programs

These application programs assist users in applying different analysis techniques; thereby, collecting useful information about certain properties of Petri nets. Some of these programs may be fully automated, while others will require an elaborated interaction with the user. Analysis programs should offer a suitable set of analysis functions to fit the user's needs. Certain of these programs will even allow the user to make formal proofs of the modeled system [Ager73]. However, the most common type of analysis programs allow users to examine Petri nets and make informal experiments with their behavior.

1.3 A discussion of our work

Much of the early research on Petri nets concentrated on the area of analysis, but not on the modeling of systems. Even researchers that dealt with modeling rarely had a computer simulation system with a graphics display. Not until recently did researchers started looking at the simulation aspect of modeling systems by Petri nets. As a result, over the last decade, there has been a rapid increase in interest over the development of Petri net tools. [Feld86] gives a good "quantitative" description of graphical Petri net tools. The purpose of our work is to design and implement a Petri net tool package to

create, simulate, and analyze Petri nets. This designer has no knowledge that the Petri net tool is developed in an Object-Oriented manner. We chose the Interlisp-D environment that runs on the Xerox[1] 1100 series Workstation [Xerox83] along with the object-oriented programming language LOOPS [Bobrow83] for our implementation of a Petri net tool package. We discuss the LOOPS programming environment in more detail in the following section. The Petri net tool package that has been developed consists of a graphical net editor, a net simulator, and an analyzer. A user models the system of interest as a Petri net by using the editor and observes the properties and behavior of that system by running the net simulator. The graphical editor assists users in constructing and modifying Petri nets. Users work directly with the graphical representation of a Petri net and can see the intermediate results of their work on the screen. The simulator interprets the resulting Petri net and simulates it directly. Program simulation is considered an important part of net tools because users can observe the behavior of systems that are modeled. The net analyzer provides useful statistical information about properties of the system being modeled. This information is collected from a reachability tree that is implemented in the tool package. The properties of Safeness, Boundedness, and Conservation are looked at by the analyzer. We will talk about the reachability tree in the next chapter.

1.4 LOOPS and Object-Oriented programming

The object-oriented style of programming is ideal for problems in program simulation where collections of things that interact with one another have to be represented. This style of programming has also been advocated for applications in graphics, simulation and modeling, systems prototyping, and AI environments. LOOPS is a knowledge

[1]Xerox is a trademark of Xerox Corporation.

programming system that adds object-access and rule-oriented paradigms to the procedure-oriented paradigm of Interlisp. The integration of LOOPS into the Interlisp environment provides access to the standard procedure-oriented programming of Lisp, along with the extensive environmental support of the Interlisp-D system. Certain major ideas surface when talking about object-oriented programming: objects, classes, message sending, and inheritance. Objects are considered the most primitive elements of object-oriented programming. Objects combine the attributes of procedure and data, and are capable of performing computations and saving local state. In many object-oriented languages, objects are divided into the major categories of classes and instances. A class in object-oriented programming corresponds to a type in procedural programming languages. For example, the class 'Place' may be a description of the structure and behavior for instances such as place1 and place2. A class determines the structure and behavior of object instances. Therefore, object instances that are similar in structure should logically belong to the same class. LOOPS supports both class variables and instance variables. Every instance in LOOPS belongs to exactly one class. The methods and structure of an instance is determined by its class. Class variables are used to hold information that is shared by all instances of the class. Instance variables store information that is specific to a particular instance. The means of communication between objects (classes and instances) is via sending message. Message sending is a form of indirect procedure call. A message to an object contains the selector and other parameters that are needed to accomplish a task. The selector in a message specifies the kind of operation to be performed. This style of communication allows each class to implement its own way of responding to a message. Objects respond to messages through methods that are used to perform operations. Class inheritance allows the creation of objects that are similar to each other but not identical. The

inheritance network in LOOPS is arranged in a lattice. When a class is placed in the lattice, that class inherits all the variables and methods from its superclasses. The LOOPS programming environment allows users to reorganize the inheritance network by providing an interactive graphics browser. Examples of changes that can be made to the network through the browser are adding and deleting classes, renaming classes, and rerouting inheritance paths.

CHAPTER 2
PETRI NETS

2.1 A brief history of Petri nets

Petri nets originated with the doctoral dissertation of Carl Adam Petri [Petri62]. Petri formulated the basis for a theory of communication between the asynchronous components of a computer system. His dissertation was mainly a theoretic development of the basic concepts from which Petri nets developed. A new model of information flow for systems resulted from this thesis.

A.W. Holt and other researchers at the Information System Theory Project of Applied Data Research, Inc. (ADR) developed much of the theory, notations, and representation of Petri nets. Their work was later published in a final report for that project [Holt68]. This project showed how Petri nets could be applied to the modeling and analysis of systems with concurrent components. The theory of "systemics', as it was called, was developed by this group, and this theory dealt with the representation and analysis of systems and their behavior.

Professor Jack B. Dennis directed the Computation Structures Group to a considerable amount of research and publication on Petri nets. This group has been a productive source of research and literature in the field. Two important conferences on Petri net were held by the Computation Structures Group: the Project MAC Conference on Concurrent Systems and Parallel Computation [Dennis70] and the Conference on Petri nets and Related Methods in 1975 at M.I.T. Both of these conferences have helped to disseminate results and approaches in Petri net theory. Project MAC has since changed its name to the Laboratory for Computer Science.

An advanced course on Petri nets was organized in 1979 in Hamburg, West Germany. The intent of this course was aimed at systematizing the existing volume of knowledge on Petri nets, and making it more accessible to a wider audience of computer scientists who are interested in the subject. This course initiated a lot of new research into the applications and theory of Petri nets. Another advanced course was later held in 1986 in Bad Honnef, West Germany, where the most important current developments in Petri nets were presented. This two courses have helped to clarify the basic philosophy underlying the Petri net approach. Since the Hamburg course, an annual European Workshop on Application and Theory of Petri Nets has been organized. A Petri Net Newsletter is now published three times a year by the Special Interest Group on Petri Nets and Related System Models of the Gesellschaft fur Informatik. The Advances in Petri Nets within the Lecture Notes in Computer Science series is also published annually by Springer-Verlag publishing company.

Theoretical journals on Petri nets can be found in Theoretical Computer Science, Journal of Computer and System Sciences, Information and Control, and Journal of the ACM. Papers that are published in these journals are often made available as technical reports first. Another important source of Petri net research is the Institut fur Informationssystemforschung of the Gesellschaft fur Mathematik und Datenverarbeitung in Bonn, West Germany.

2.2 Definitions

A Petri net is composed of four parts; a set of places P, a set of transitions T, an input function I, and an output function O. The input function I is a mapping from a transition t_j to a set of places $I(t_j)$, known as the input places of the transition. The output function O

maps a transition to a set of places $O(t_j)$, known as the output places of the transition. A place p_i is an input place of a transition t_j if $p_i \varepsilon I(t_j)$; p_i is an output place of transition t_j if $p_i \varepsilon O(t_j)$. The input and output functions relate transitions and places. Hence, the structure of a Petri net is defined by its places, transitions, input function, and output function. An example of a Petri net structure is given in figure 2.1.

Definition 2.1

A Petri net structure, C, is a four-tuple, C = (P,T,I,O). $P = \{p_1, p_2, ..., p_n\}$ is a finite set of places, $n \geq 0$. $T = \{t_1, t_2, ... , t_n\}$ is a finite set of transitions, $m \geq 0$. The set of places and the set of transitions are disjoint, $P \cap T = 0$. I: T -> P is the input function, a mapping from transitions to sets[2] of places. O : T -> P is the output function, a mapping from transitions to sets of places. The set of input places of a transition t is given by $I(t) = \{p \mid (p,t) \varepsilon A\}$, where A is the set of arcs in C. The set of output places of a transition t is given by $O(t) = \{p \mid (t,p) \varepsilon A\}$.

C = (P, T, I, O)

P = {p1, p2, p3, p4, p5}

T = {t1, t2, t3, t4}

I(t1) = {p1}	O(t1) = {p2}
I(t2) = {p2}	O(t2) = {p1, p3}
I(t3) = {p3, p4}	O(t3) = {p5}
I(t4) = {p5}	O(t4) = {p4}

Figure 2.1 A Petri net structure

[2]In general, the input and output functions map transitions to bags of places.

A Petri net structure consists of two basic components: places and transitions. Corresponding to these components, a Petri net graph has circles that represent places and bars that represent transitions. Directed arcs, indicated by lines with arrows, represent the input and output functions and connect the places and the transitions. Some arcs are directed from places to transitions, while others are directed from transitions to places. If an arc is directed from node i to node j, either from a place to a transition or from a transition to a place, we say that node i is an input to node j and node j is an output of node i. A Petri net is a directed graph[3] because all the arcs in the graph are directed. In addition, all the nodes in the graph can be partitioned into two sets such that each arc is directed from an element of one set to an element of the other set. Therefore, Petri net structures can be represented as bipartite directed graphs in the form of Petri net graphs. The representation of a Petri net as a graph in pictorial form is common practice in Petri net research. The Petri net graph can be used to model the static properties of a system just as a flowchart would represent the static properties of a computer program. As opposed to the static properties that are represented by the graph, a Petri net also has dynamic properties that result from its execution. We show a Petri net graph in figure 2.2 that corresponds to the Petri net structure of figure 2.1.

Definition 2.2

A Petri net graph, G, is a bipartite directed graph, G = (V,A), where $V = \{v_1, v_2, \ldots, v_s\}$ is a set of vertices, and $A = \{a_1, a_2, \ldots, a_r\}$ is a set of directed arcs, $a_i = (v_j, v_k)$,

[3] In general, Petri nets are multigraphs because more than one arc is allowed between any place and transition. For the purpose of this implementation, we allow only one arc between places and transitions.

with v_j, v_k ε V. The set V can be partitioned into two disjoint sets, P and T, such that $V = P \cup T$, $P \cap T = \emptyset$. For each directed arc, if $a_i = (v_j,v_k)$, then either $v_j \; \varepsilon \; P$ and $v_k \; \varepsilon \; T$ or $v_j \; \varepsilon \; T$ and $v_k \; \varepsilon \; P$.

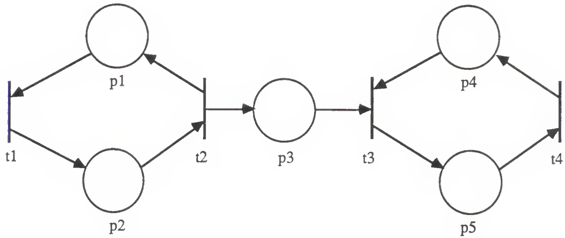

Figure 2.2 A Petri net graph equivalent to the Petri net structure of figure 2.1

A Petri net which has tokens is called a marked Petri net. Tokens can only be assigned to the places of a Petri net. The distribution of tokens in a marked Petri net defines the state of the Petri net and is called its marking. A marking μ is an assignment of tokens to the places of a Petri net. On a Petri net graph, tokens are represented by small solid dots • inside the circles that represent places of the net. Tokens are used to define the execution of a Petri net. The number of tokens in any place of the net may change as a result of executing the Petri net. Tokens may move from one place to another place during the execution of a Petri net. A Petri net C = (P,T,I,O) with a marking μ becomes the marked Petri net, M = (P,T,I,O,μ). Figure 2.3 is a graph representation of the marked Petri net for the structure of figure 2.1.

Definition 2.3

A marking μ of a Petri net C = (P,T,I,O) is a function from the set of places P to the nonnegative integers N, μ: P -> N.

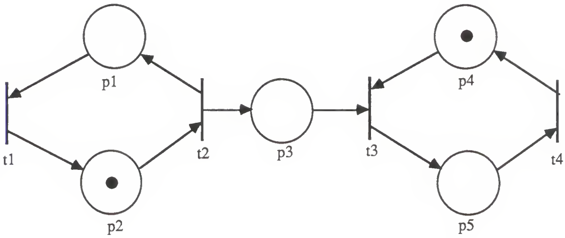

Figure 2.3 A marked Petri net with the same structure as figure 2.2

The behavior of a Petri net is given by sequences of enabled transitions that are fired. A transition becomes enabled if each of its input places possesses a token. A transition fires by removing tokens from its input places and creating new tokens in its output places. A token is removed from each input place that has an arc pointing to the transition and is placed into each output place that has an arc leading to it from that transition. Different transitions may become enabled in different markings. The execution of a Petri net is controlled by the number as well as the distribution of tokens in the Petri net. Tokens in the input places which enable a transition are called the enabling tokens. Furthermore, tokens are indivisible. This means that a token cannot be removed from a place by two different transitions at the same time. Transition t2 is enabled in figure 2.3. If t2 fires, a token is deposited into each of the output places, p1 and p3. Figure 2.4(a) is the marked

Petri net that results from firing transition t2. In this new marking, transitions t1 and t3 become enabled. On firing transition t3, a different marking occurs. The result is shown in figure 2.4(b).

Definition 2.4

A transition t_j ε T in a marked Petri net C = (P,T,I,O) with marking μ is enabled if for all p_i ε I, $\mu(p_i) > 0$. $\mu(p_i)$ gives for each place p_i in a Petri net the number of tokens in that place.

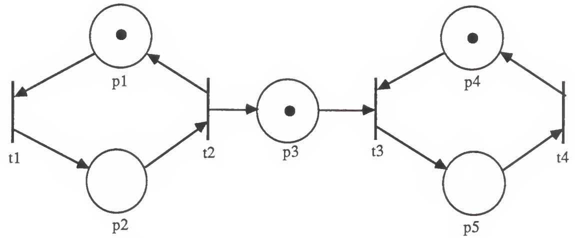

Figure 2.4(a). Marked Petri net with t3 enabled

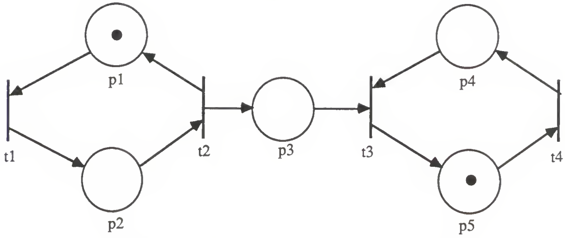

Figure 2.4(b). Marked Petri net after firing t3..

A state of a Petri net is defined by its marking. The firing of a transition represents a change in the state of a Petri net as indicated by a change in the marking of the net. The state space of a Petri net with n places is the set of all markings, that is, N^n. This change of state is defined by the change function ∂; this is also called the next-state function.

Definition 2.5

The next-state function, ∂: $N^n \times T \rightarrow N^n$, for a Petri net $C = (P,T,I,O)$ with marking μ and transition $t_k \in T$ is defined if and only if $\mu(p_i) \geq 1$ for all $p_i \in I(t_k)$.

If $\partial(\mu, t_k)$ is defined, then $\partial(\mu, t_k) = \mu'$;

where, $\mu'(p_i) = \mu(p_i) - 1$ for all $p_i \in I(t_k) - O(t_k)$,

$\mu'(p_i) = \mu(p_i) + 1$ for all $p_i \in O(t_k) - I(t_k)$,

otherwise $\mu'(p_i) = \mu(p_i)$.

Two sequences can result from the execution of a Petri net; the sequence of markings (μ^0, μ^1, μ^2, ...) and the sequence of transitions which were fired (t_{j0}, t_{j1}, t_{j2}, ...).

The set of all markings which are reachable from a Petri net C with a marking μ is called the reachability set $R(C,\mu)$. A marking μ' is in $R(C,\mu)$ if there is a sequence of transition firings which will change marking μ into marking μ'.

Definition 2.6

The reachability set $R(C, \mu)$ for a Petri net $C = (P, T, I, O)$ with marking μ is the smallest set of markings defined by :

1) $\mu \varepsilon R(C, \mu)$,

2) If $\mu' \varepsilon R(C, \mu)$ and $\mu'' = \partial(\mu',t_j)$ for some $t_j \varepsilon T$, then $\mu'' \varepsilon R(C, \mu)$.

A place in a Petri net is safe if the number of tokens in that place never exceeds one. A Petri net is safe if all the places in that net are safe. Safeness is a very important property when a Petri net is used to model a real hardware device. The number of tokens in a safe place is either 0 or 1. This property is related to the original definition of Petri nets which was given in terms of events and conditions. In that definition, a condition was represented by a place. If a token exists in a place, the condition is said to hold. Since a condition should either hold or not hold, the presence or absence of a token is sufficient to denote either condition. Therefore, no more than one token is needed in any place of the net.

Definition 2.7

A place $p_i \varepsilon P$ of a Petri net $C = (P,T,I,O)$ with initial marking μ is safe if for all $\mu' \varepsilon R(C,\mu)$, $\mu'(p_i) \leq 1$.

If there is a bound of only one token in every place of the net, the net is said to be safe. A place is bounded if it is k-safe for some k; a place is k-safe or k-bounded if the number of tokens in that place never exceeds an integer k. A Petri net is bounded if all the places in that net are bounded. A bounded Petri net can be implemented in hardware as a counter.

Definition 2.8

A place $p_i \varepsilon P$ of a Petri net $C = (P,T,I,O)$ with an initial marking μ is k-safe is for all $\mu' \varepsilon R(C,\mu)$, $\mu' (p_i) \leq k$.

The property of conservation is important if Petri nets are used to model resource allocation systems. In these systems, tokens may be used to represent the resources and it is important to show that tokens are neither created nor destroyed. One simple way to preserve the conservative property is to require that the total number of tokens in the net remain constant. A Petri net is said to be conservative if the number of tokens in the net stays the same at all times. This may further be interpreted as the number of inputs of each fireable transition being equal to the number of outputs of that transition.

Definition 2.9

A Petri net $C = (P,T,I,O)$ with an initial marking μ is strictly conservative if for all $\mu' \varepsilon R(C,\mu)$,

$$\sum_{p_i \varepsilon P} \mu'(p_i) = \sum_{p_i \varepsilon P} \mu(p_i)$$

2.3 The Analysis problem

Several major techniques for analyzing Petri nets have been suggested in the literature. Two of these techniques provide solution mechanisms for many of the analysis problems in Petri nets. One technique makes use of a reachability tree, while the other involves the use of matrix equations. Since the reachability tree approach was taken for the implementation of our thesis, we will only mention this approach here. The technique of using a reachability tree involves finding a finite representation for the reachability set of a Petri net. Since the reachability set of a marked Petri net is often infinite, the reachability tree may also be infinite. Even a Petri net with a finite reachability set can have an infinite tree. The reachability tree has nodes that represent markings of the Petri net and arcs that represent possible changes in state that result from the firing of transitions [Karp69]. The reachability tree represents all the possible sequences of transition firings. In order to limit the size of an infinite tree to a finite representation, many markings of the tree have to be mapped into the same node.

Several conditions help to reduce an infinite tree to a finite representation. The first condition occurs when nodes have markings in which no transitions are enabled. These markings are saidto be dead, and no new markings are generated in the tree by this type of nodes. These dead markings are also called terminal nodes. Another class that is helpful

in limiting the size of the tree involves markings which have previously appeared in the tree. These repeated markings are known as duplicate nodes. Successors of a duplicate node do not have to be considered because all these successors may be generated from the first occurrence of the marking in the tree. One final means is used to reduce the reachability tree to a finite representation. When the number of tokens in a place of the net becomes too large, a set of states can be collapsed into a single node by ignoring the number of tokens in that place. This situation occurs because a new marking is generated that is greater than or equal in every component to some other marking along the path of the tree from the root to that new marking. The set of states between these two markings can be repeated indefinitely, thereby allowing a component of the newly generated marking to increase without bound. Nodes whose tokens at a particular place can become arbitrarily large are called interior nodes. Since the number of tokens at this type of nodes may increase without bound, a special symbol is used to denote this case. The symbol is represented by a w, and it stands for a value which has no limit. To show an example of generating the reachability tree for a Petri net, we will consider the marked Petri net of figure 2.5. This Petri net has the initial marking (0, 1, 0, 1, 0). This initial marking becomes the root of the tree. In this marking only transition t2 is enabled. Firing t2 yields the marking (1, 0, 1, 1, 0).

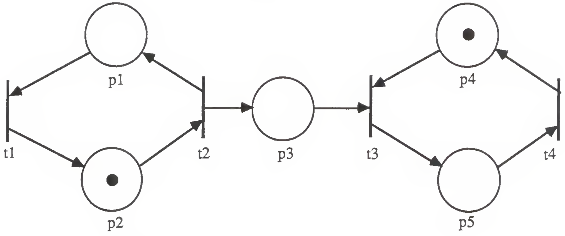

Figure 2.5. Initial marking of Petri net.

Two transitions, t1 and t3, are enabled in this new marking. Firing the enabled transitions gives two new markings, (0, 1, 1, 1, 0) and (1, 0, 0, 0, 1). A new node is defined in the reachability tree for each marking which results from firing an enabled transition (see figure 2.6). Since all the components in the marking that results from firing t1 is either greater than or equal to the root marking, $(0,1,1,1,0) \geq (0,1,0,1,0)$, we can replace the third component of this new marking by a w. The new marking (0, 1, w, 1, 0) becomes an interior node in the tree. The presence of a w in marking (0, 1, w, 1, 0) reflects the fact that the sequence t2-t1 can be fired an arbitrary number of times. We can also view the presence of a w as producing an infinite number of markings. The resulting reachability tree after performing these first two steps is shown in figure 2.7.

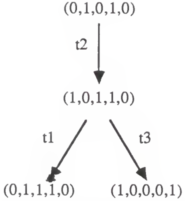

Figure 2.6. Reachability tree before replacing the third component of (0,1,1,1,0) by w.

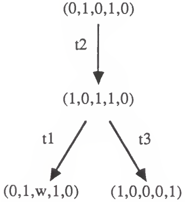

Figure 2.7 The first two steps in building the reachability tree.for the Petri net of figure 2.5

From the marking (0, 1, w, 1, 0) we can fire t2 to give the marking (1, 0, 2, 1, 0). Since (1, 0, 2, 1, 0) $\geq$ (1, 0, 1, 1, 0) we can again replace the third component by w. This marking, (1, 0, w, 1, 0) is also an interior node. Repeating this process new markings are added to the tree until all nodes become duplicate, terminal, or interior nodes. The complete reachability tree that is generated for the Petri net of figure 2.5 is shown in figure 2.8. Duplicated nodes are shown as underlined and bold face text. Interior nodes are in bold face only.

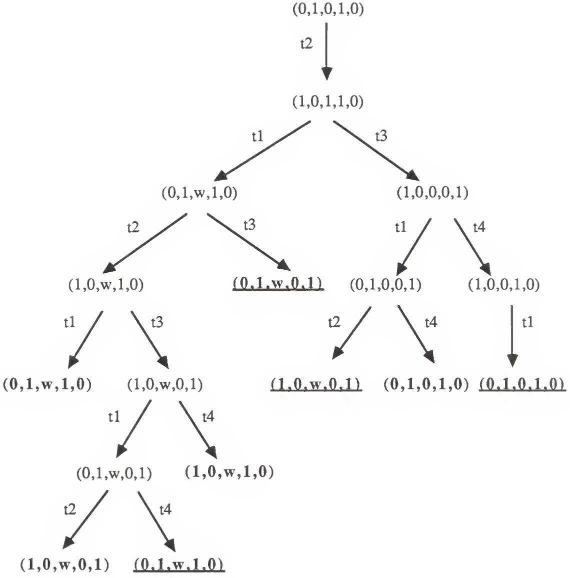

Figure 2.8 The reachability tree for the Petri net of figure 2.5.

CHAPTER 3

A PETRI NET TOOL PACKAGE

3.1 An Overview of PETRISYS

The formal definition of Petri nets provides a basis for theoretical work on the subject. However, graphical representations of Petri nets are often more useful in illustrating the concepts of Petri net theory. Therefore, a computer system for editing, simulating, and analyzing Petri nets has been implemented in the Computing and Information Science department at Kansas State University. This system is named PETRISYS. PETRISYS provides an interactive environment whereby a user first designs the system to be modeled using a Petri net graphical editor, and then he is able to check for properties of the modeled system by making use of the PETRISYS simulator and analyzer. A display of a Petri net marking can be represented in PETRISYS and the behavior of that net simulated graphically. PETRISYS models a restricted version of Place/Transition Systems (called P/T-systems, for short). The main difference between systems that are modeled by PETRISYS and the P/T systems is places in nets modeled by PETRISYS are allowed to hold more than one token, but arcs can have only weight one. Nets in PETRISYS are edited graphically in a window through the use of a menu. A mouse is used as the input device, thereby freeing the user from having to enter text as input. A simulation program will combine the editor-created net with an interpretation and simulate the net directly. The simulation of Petri nets may be done either interactively or automatically. The user is able to watch the dynamic progress of the simulation and interact with the net at any point during its execution. Interactive simulation is primarily used during the design of a complicated system or for demonstration purposes. Automatic

simulation can be used when the simulation to be performed is lengthy. The user is thus freed to do some other useful work. Another important use of automatic simulation is in rapid prototyping.

3.2 Operating PETRISYS

PETRISYS is available on the Xerox workstations of the Computing and Information Science Department at Kansas State University. We assume that the reader is familiar with these systems. At the beginning of a session two windows appear on the screen: the 'Prompt' window and the 'TTY' window. The session begins by setting the current time in the TTY window. This window should appear on the left side of the screen and has the cursor in it. The format for setting the time is "SETTIME "MM-DD-YY HOUR:MINUTES". See figure 3.1 for an example. We recommend the user to reshape the TTY and Prompt windows to a smaller size. To save more space on the screen one also may remove the clock and history icons by closing them. The PETRISYS program resides in "{DSK}<LISPFILES>TAN>". All the files in this directory appears in the filebrowser window at this time (see figure 3.2). The file 'MENUBITS' is selected by pointing the mouse to it and clicking on the left button. Next, the file 'PROJECT' is selected by clicking the middle mouse button. The middle button selects a file in addition to the other files which may have already been selected. Both files should have arrows to their side at this time. 'MENUBITS' contains the bitmaps for icons that are used by the application program. 'PROJECT' is the filename that the application program resides in. Select load from the filebrowser menu at this point. Loading takes approximately five to ten minutes. When the loading is done, you are ready to start PETRISYS. Shrink the filebrowser window and position it under the TTY window next to the LOOPS icon. The

user starts a session with the function 'PETRI' in the TTY window. The word PETRI has to be enclosed between parenthesis without quotes (see figure 3.3). This initializes all class variables and instance variables in the application program to their default values. A hierarchy of classes is also automatically created in LOOPS. The graphical interface of PETRISYS appears after a few seconds. This main interface consists of an editing window, a main menu of operation selections, and a prompt window for displaying messages to the user. You can now start to draw a net in the editing window. Section 3.3

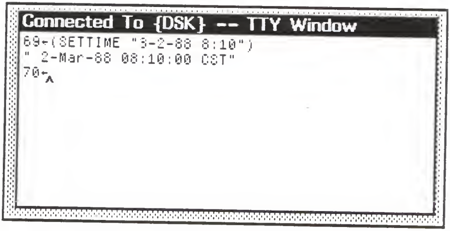
Connected To {DSK} -- TTY Window
69←(SETTIME "3-2-88 8:10")
" 2-Mar-88 08:10:00 CST"
70←

Figure 3.1

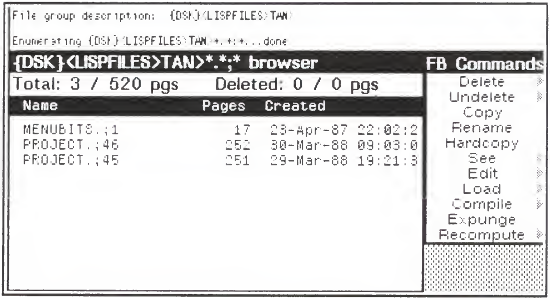
File group description: {DSK}<LISPFILES>TAN>
Enumerating {DSK}<LISPFILES>TAN>*.*;*...done
{DSK}<LISPFILES>TAN>*.*;* browser
FB Commands
Total: 3 / 520 pgs
Deleted: 0 / 0 pgs
Name
Pages
Created
MENUBITS.;1 17 23-Apr-87 22:02:2
PROJECT.;46 252 30-Mar-88 09:03:0
PROJECT.;45 251 29-Mar-88 19:21:3
Delete
Undelete
Copy
Rename
Hardcopy
See
Edit
Load
Compile
Expunge
Recompute

Figure 3.2

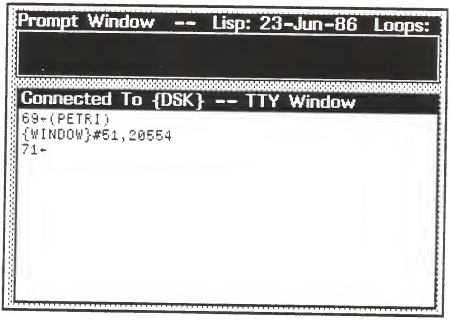
Prompt Window -- Lisp: 23-Jun-86 Loops:
Connected To {DSK} -- TTY Window
69←(PETRI)
{WINDOW}#51,20554
71←

Figure 3.3

describes the editing process. A PETRISYS session is ended by clicking on the broom option followed by the closing of the editing window. To close the editing window, the mouse is moved into the window and the right button is held down. When a popup menu appears, the 'close' option is chosen. The prompt should now return to the TTY window.

3.3 The PETRISYS Graphical Net Editor

A two-dimensional net model is created and edited with the PETRISYS graphical editor. Net components (places, transitions, arcs, and tokens) can be added (drawn), deleted (erased), and inspected. This graphical net editor provides the normal requirement for a good user interface; it offers a menu for command selection. The main interface to users consists of an editing window with an attached menu and a message window (see figure 3.4). All the options for editing, simulating, and analyzing Petri nets are represented in the main menu. Certain options such as place, transition, arc, token, and clear screen are represented as icons, while the other options are given as names of the operations. The main menu of PETRISYS is shown in figure 3.5.

The process of drawing net components involve selecting the appropriate option from the menu. A message is given in the message window to prompt the user for a position that the net component will be drawn at (see figure 3.6).

The 'arc' option can be used to draw arcs between places and transitions. When the user selects this option, he is prompt twice; once for the location that the arc will start from and another time for the location that the arc will end at. The arc is then drawn from the starting location to the ending location (see figure 3.7). A check is made to ensure that no arc already exists between the two objects.

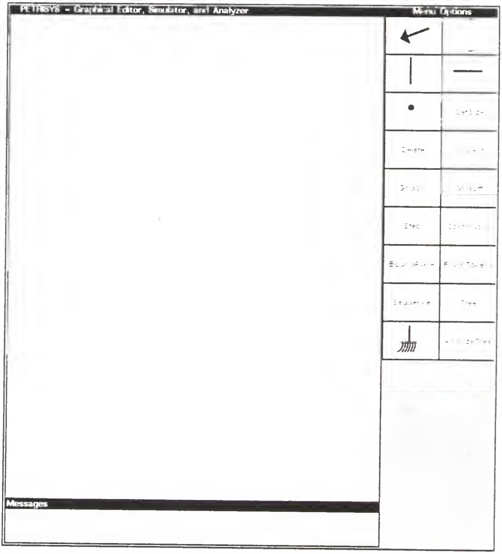
PETRISYS - Graphical Editor, Simulator, and Analyzer
Menu Options
Messages

Figure 3.4

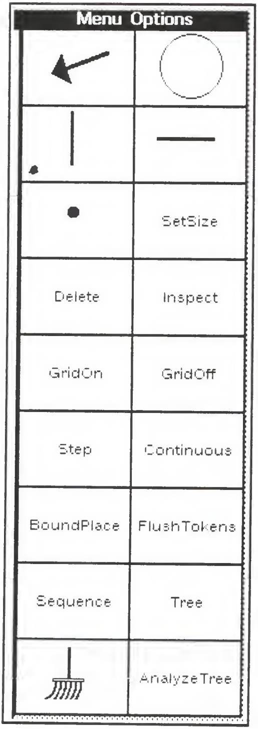
Menu Options
SetSize
Delete
Inspect
GridOn
GridOff
Step
Continuous
BoundPlace
FlushTokens
Sequence
Tree
AnalyzeTree

Figure 3.5

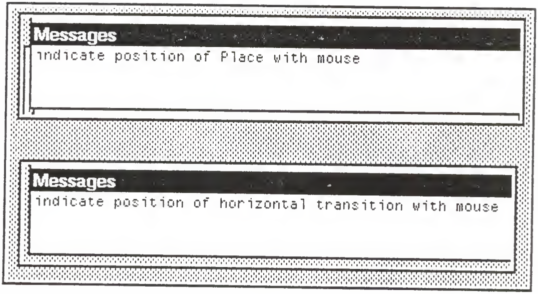
Messages
indicate position of Place with mouse
Messages
indicate position of horizontal transition with mouse

Figure 3.6

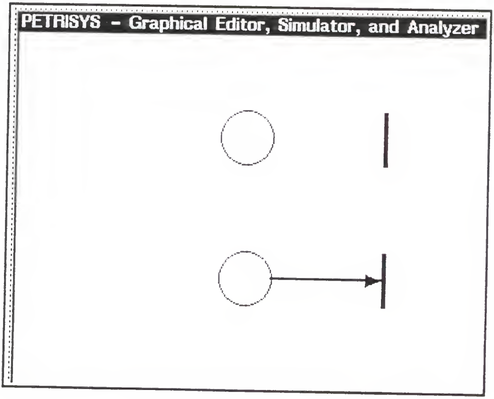

Figure 3.7

A token can be drawn by first choosing the 'token' icon from the menu, shown as a solid dot, and indicating the place to deposit this token with the mouse. Tokens will only go into places. Furthermore, tokens are automatically arranged in each place if the count is less than or equal to four. If the number of tokens in a place gets more than four, the number of tokens in that place is shown as a digit to represent this number (see figure 3.8).

The 'SetSize' option allows the user to choose the size of net components that he wants to draw on the screen. A popup menu with three choices appears after this option is selected. The allowable options are 'small', 'medium', and 'large'. The SetSize option is limited in that it can only be selected at the beginning of each editing session. This means that users have to decide the size of net components before starting to draw a new Petri net.

The 'Delete' option allows users to remove places and/or transitions from the graphical display. Whenever a place or a transition is removed from the net, any connecting arc(s) and labels will be deleted as well (see figure 3.9). This feature prevents the user from disrupting the syntactic rules of Petri net theory. Any editing operation in PETRISYS preserves the syntactic correctness of the Petri net. At this time arcs cannot be deleted independent of a place or a transition.

PETRISYS automatically labels places and transitions as they are drawn in the editing window. Places are named beginning with the letter 'p' followed by a number, whereas transition names begin with a 't' followed by a number (see figure 3.10). The object label is placed beside the object after the user has selected a position with the mouse.

Any place is allowed to have a bound on the number of tokens that can reside in that place. Users can set this bound by choosing the 'BoundAPlace' option from the main menu. The place to put a bound will be prompt for. The default bound for all places is 10.

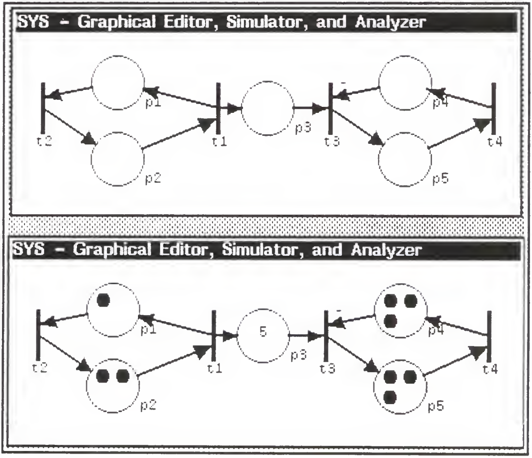
SYS - Graphical Editor, Simulator, and Analyzer
p1
p2
p3
p4
p5
t1
t2
t3
t4
SYS - Graphical Editor, Simulator, and Analyzer
5
p1
p2
p3
p4
p5
t1
t2
t3
t4

Figure 3.8

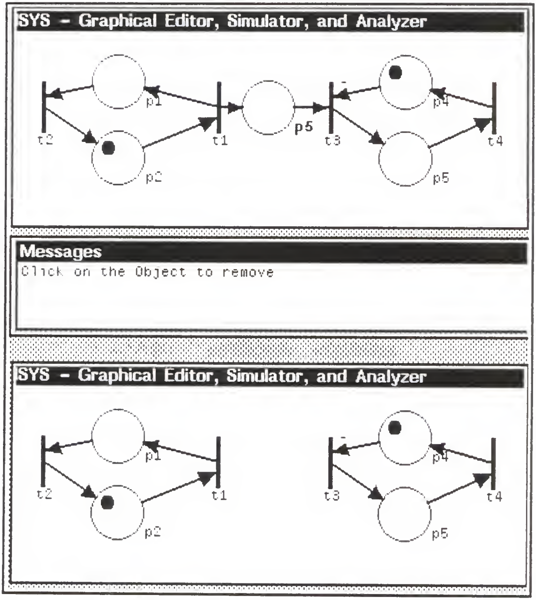
SYS - Graphical Editor, Simulator, and Analyzer
p1
p5
p4
t2
t1
t3
t4
p2
p5
Messages
Click on the Object to remove
SYS - Graphical Editor, Simulator, and Analyzer
p1
p4
t2
t1
t3
t4
p2
p5

Figure 3.9

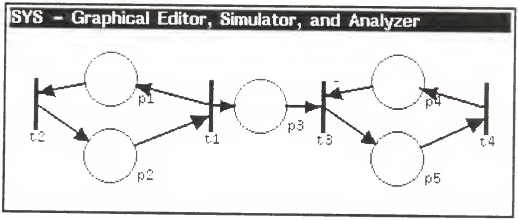
SYS - Graphical Editor, Simulator, and Analyzer
p1
p2
p3
p4
p5
t2
t1
t3
t4

Figure 3.10

A grid is available in PETRISYS for the user to arrange objects during editing. The user is able to carefully align objects in the editing window when the grid is turned on. This grid can be turn on by selecting the appropriate option in the menu. A click on the 'Grid On' option will turn on the grid. To disable the grid simply select the 'Grid Off' option from the menu.

The 'FlushTokens' option erases all the tokens that are present in the current net. User may find this option useful when it is necessary to simulate a net with many different markings. The top diagram of figure 3.11 shows the Petri net before a 'FlushToken' command is issued. The resulting Petri net is shown in the bottom diagram of the same figure.

The 'Tree' option informs PETRISYS to generate a reachability tree internally for the current Petri net model in the editing window. A window opens and markings of the reachability tree are displayed. Click on the 'Sequence' option to see the sequence of transitions that fired in the process of generating this tree. We show the results of these operations by an example in the last section of this chapter.

The 'Clear Screen' option, represented by the broom icon, destructively erases the current Petri net from the screen. A new Petri net can now be drawn. A side effect of using this option is that all instance variables as well as class variables are reinitialized to nil.

3.4 The PETRISYS Simulator

Users can simulate nets that are modeled by PETRISYS in real-time. Nets in PETRISYS are executed in either 'step' mode (interactive) or 'continuous' mode (automatic). One enabled transition fires each time the 'step' option is selected from the

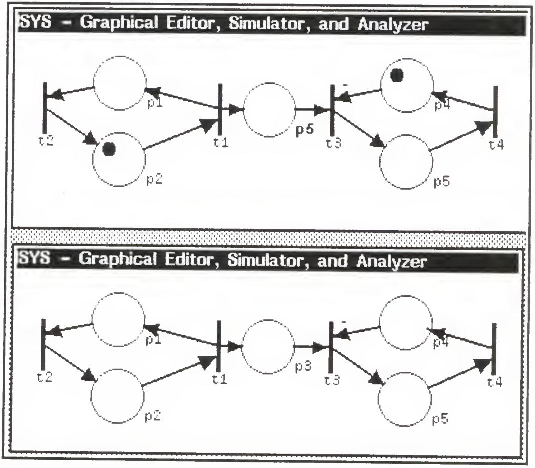
SYS - Graphical Editor, Simulator, and Analyzer
p1
p2
p4
p5
p5
t1
t2
t3
t4
SYS - Graphical Editor, Simulator, and Analyzer
p1
p2
p3
p4
p5
t1
t2
t3
t4

Figure 3.11

menu. In step mode, users have the opportunity to observe the firing of transitions one step at a time. Users can observe net simulation at machine execution speed in continuous firing mode. In this mode, the next transition to fire is chosen at random from among all transitions that are enabled at that time. Net execution continues until either no more transitions are enabled or the user halts execution. Transition firing is depicted by actually moving tokens from the input places to the output places of the enabled transition. Enabled transitions are highlighted just before they fire to give users a better feel of which transition actually fired. You can pause the simulation at any time by clicking on the LEFT mouse button. After net execution halts, users can continue execution from that point, at either step or continuous mode, or stop execution altogether. The PETRISYS simulator program is an interpretator program. Thus, Petri net components can be changed and simulated right away. The sequence of transitions that is produced when a Petri net executes is saved by PETRISYS. This sequence can be seen by clicking on the 'Sequence' option. The two options that appear in a popup menu are 'ShowSequence' and 'EraseSequence'. 'ShowSequence' opens a window to display the sequence of transitions that just fired (see figure 3.12). 'EraseSequence' removes the firing sequence from memory and closes the window at the same time.

3.5 The PETRISYS Syntax Checker

PETRISYS has a syntax-checker that makes sure syntactic structures are maintained in the Petri net. The syntax-checker is not explicitly visible but rather lurks in the background whenever PETRISYS is in used. The syntax-checker is mostly responsible for checking 'correct' net structure when the editor is in use. Anytime the layout of a Petri net in PETRISYS is modified, the syntax checker makes sure that the diagram reflects the

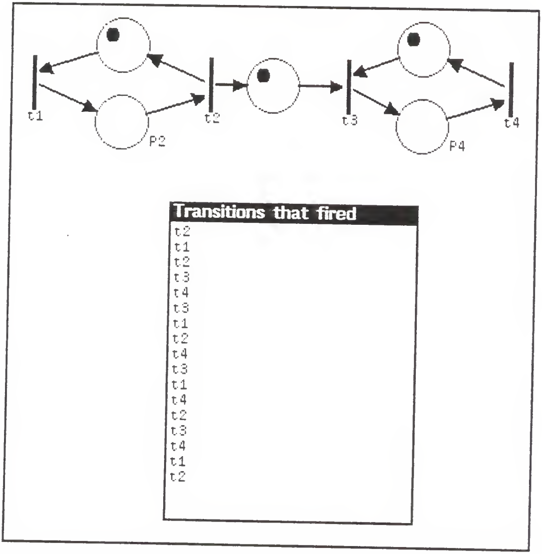
t1
P2
t2
t3
P4
t4
Transitions that fired
t2
t1
t2
t3
t4
t3
t1
t2
t4
t3
t1
t4
t2
t3
t4
t1
t2

Figure 3.12

underlying structure. An example of such a check happens when a user tries to connect an arc between two transitions without any intervening place. The user will be informed of the syntactic mistake, but no arc will be drawn between the transitions. Another check occurs when a node, either places or transitions, is deleted from the diagram. Arcs that connect the node to other nodes are deleted and no arcs are left dangling. This kind of structural control is quite popular in computing science, and is similar to the use of syntax-directed editors for programming languages. It frees a user from the tedious job of trying to maintain the correct syntax for the system at all times. However, semantic checks are the responsibility of the user. One example is that arcs may be drawn through a place if that place is on the path between two nodes that the arc connects (see figure 3.13).

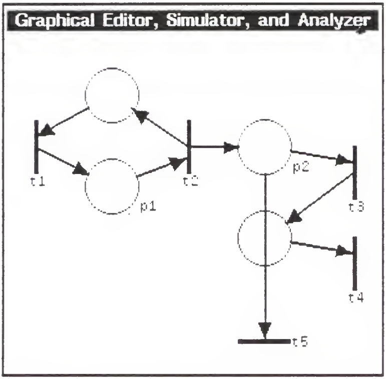

Figure 3.13

3.6 The PETRISYS Analyzer

Modeling a system by a Petri net in itself is not sufficient or useful enough for all purposes. A user cannot learn much by just watching the simulation. To gain important insights into the system's behavior, it is necessary to analyze the modeled system. Petri net models of systems are checked for the properties of Safeness, Boundedness, and Conservation by the PETRISYS analyzer. For example, if a user is modeling a resource allocation system, he might be interested whether the resources being modeled are preserved in the Petri net. The property of conservation is required in this case. If the system which is modeled is found to be conservative, resources in this system are known to be preserved. The PETRISYS Analyzer uses the reachability tree technique of analyzing systems. The 'AnalyzeTree' option in the main menu displays the properties of Safeness, Boundedness, and Conservation for the reachability tree that has been produced. The reachability tree approach of analysis has its limitations. For example, the general problems of reachability and liveness cannot be solved using the reachability tree approach. We did not try to be exhaustive in the properties that are analyzed, but instead tried to show certain properties that can be analyzed.

3.7 An example of modeling with Petri nets

We illustrate the modeling of systems using Petri nets by considering the producer/consumer problem. The producer/consumer problem depicts an environment whereby a producer produces items and deposits them into a buffer for a consumer to consume. This is a problem in which the producer and consumer has to coordinate with each other. The consumer process cannot consume an item if no such item exists in the buffer. Therefore the consumer has to know if an item exists in the buffer. The

availability of an item to be consumed is indicated by the presence of a token in the buffer. As long as a token is present in the buffer, the consumer can remove one token from it at a time. On the other hand, the buffer may be limited in capacity. So the producer can only produce as many items as the buffer can hold. A Petri net that models the producer/consumer problem is shown in figure 3.14. Place p3 represents the buffer in our example. Transition t2 can now fire because it is enabled. Firing transition t2 is equivalent to the producer depositing an item into the buffer. The placing of an item into the buffer is depicted by removing a token from the place p2, and depositing a token into each of the places, p1 and p3 (see figure 3.15). Transitions that are enabled at any time during execution may represent events that can occur in a system. Whenever a net is executed different states may be reached. Several markings are produced when this happens. The set of enabling tokens at the various stages of execution during simulation can be considered as "conditions". These conditions may be used to describe the state of the system being modeled.

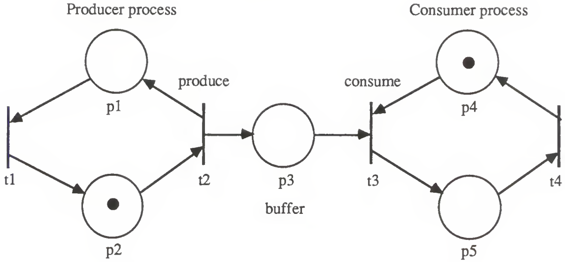

Figure 3.14 The Producer/Consumer problem modeled as a Petri net.

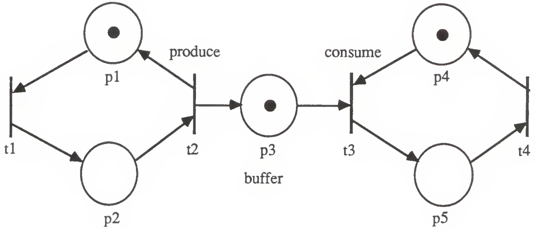

Figure 3.15 The result of depositing an item into the buffer.

Firing transition t3 at this point corresponds to the consumer consuming an item from the buffer. The occurrence of this event is represented by the removal of tokens from places p3 and p4, and the addition of a token into place p5 (see figure 3.16).

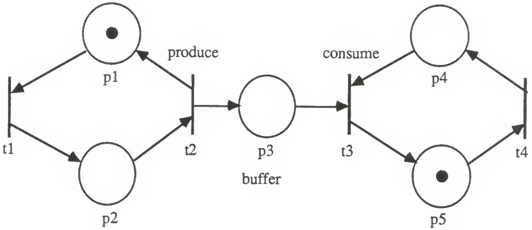

Figure 3.16 The result of consuming an item from the buffer.

The sequence of transitions that fired is shown in figure 3.17. The set of markings that the PETRISYS produces in the construction of the reachability tree for the producer/consumer problem with initial marking (0, 1, 0, 1, 0) is shown in figure 3.18. Clicking on the 'AnalyzeTree' option gives the properties of the current Petri net (see figure 3.19).

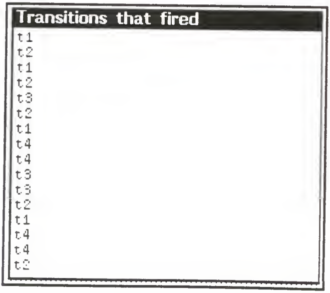

Figure 3.17

Markings in the Reachability Tree

ROOT is (0 1 0 1 0)
Fire t2 gives (1 0 1 1 0)
Fire t1 gives (0 1 w 1 0)
Fire t2 gives (1 0 w 1 0)
Fire t1 gives (0 1 w 1 0)
"Duplicate marking found!"
Fire t3 gives (1 0 w 0 1)
Fire t1 gives (0 1 w 0 1)
Fire t2 gives (1 0 w 0 1)
"Duplicate marking found!"
Fire t4 gives (0 1 w 1 0)
"Duplicate marking found!"
Fire t4 gives (1 0 w 1 0)
"Duplicate marking found!"
Fire t3 gives (0 1 w 0 1)
"Duplicate marking found!"
Fire t3 gives (1 0 0 0 1)
Fire t1 gives (0 1 0 0 1)
Fire t2 gives (1 0 w 0 1)
"Duplicate marking found!"
Fire t4 gives (0 1 0 1 0)
"Duplicate marking found!"
Fire t4 gives (1 0 0 1 0)
Fire t1 gives (0 1 0 1 0)
"Duplicate marking found!"
"End of markings for the Tree"

Figure 3.18

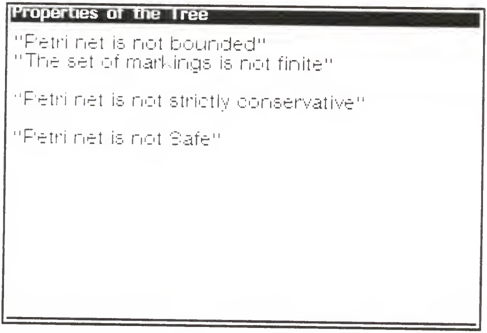
Properties of the Tree
"Petri net is not bounded"
"The set of markings is not finite"
"Petri net is not strictly conservative"
"Petri net is not Safe"

Figure 3.19

CHAPTER 4

THE PETRISYS IMPLEMENTATION

4.1 Description of the inheritance network

PETRISYS maintains an internal representation of the Petri net for the system that is modeled. Petri net components are represented by instances of classes of LOOPS objects. A hierarchy of classes is automatically created in LOOPS consisting of PetriNet, Arc, Place, Transition, HTransition, and VTransition. A browser which shows the inheritance network for the classes in PETRISYS is given in figure 4.1. The inheritance network of LOOPS is arranged in a lattice[5]. Figures 4.2 to 4.8 show the definitions of all the classes in the inheritance network. The root class of this network is called 'PetriNet'. PetriNet is also the superclass of all the other classes. All the descriptions in a class, variables and methods, are inherited by a subclass unless any of these descriptions are overridden in the subclass. This means that any variable that is defined higher in a class of the inheritance network will also appear as instances of this class. Under the PetriNet class, objects that represent the basic components of a Petri net are represented as classes; in this case, the classes 'Arc', 'Place', 'Token', and 'Transition'. Moreover, the classes 'HTransition', and 'VTransition' exist under 'Transition. Looking at the class definition for Place, shown as #$Place, PetriNet is known to be its super class. All the instance variables of a class are found under the heading IVs. Instance variables such as Bound, Center, Radius, and Region may be defined along with their default values. Under the heading CVs, values for the class variables are introduced. An example of class variables in Place is

[5]We refer to a "lattice" as a directed graph without cycles, and the lattice is allowed to havé more than one parent.

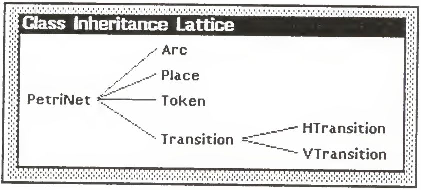
Class Inheritance Lattice
PetriNet
Arc
Place
Token
Transition
HTransition
VTransition

Figure 4.1

User Exec -- PPDefault Window

#$PetriNet

Supers

Object

IVs

ArcInst BroomInst DeleteInst HTransInst PlaceInst
Sequence SimulateInst SizeFlag TList TokenInst
VTransInst

CVs

AnalysisRegion AnalysisW ArcList ArrowY
BlankCircle Conservative EndSimulate HTransPlace
MainMenu MainW Offset PWindowRegion PlaceCount
PlaceHTrans PlaceLabels PlaceList PlaceVTrans
PromptW Safe SequenceRegion SequenceW Status
TokenList TokenSize TotalTokenCount TransCount
TransLabels TransList TreeMarkings TreeRegion
TreeW VTransPlace WindowRegion

Methods

AreYouClicked BoundAPlace CheckCurrentPath
CheckParent CheckTermination CompareObjs
CreateSequence DoAnalysis DoInspect DoSimulation
DoTree DoesRegionOverlap FindObj Fire FlushTokens
GenerateTree GetEnabledTrans ProcessArc
ProcessCleanUp ProcessDelete ProcessHTrans
ProcessPlace ProcessToken ProcessVTrans PutPrompt
RemoveSequence SeeSequence SelectGrid SetSizes
SetUpW StartSimulation UpdateSize WaitMouseDown
WaitMouseDownGrid
MessageNotUnderstood MethodNotFound SuperMethodNotFound

Figure 4.2

```
User Exec -- PPDefault Window

  #$Arc
Supers
  PetriNet
IVs
  Obj1 Obj2 PointTo PtForArrow2 PtForArrow3 PtFrom
  PtOnCircle PtOnTrans Region Type
  ArcInst BroomInst DeleteInst HTransInst PlaceInst
  Sequence SimulateInst SizeFlag TList TokenInst VTransInst
CVs
  ArrowX
  AnalysisRegion AnalysisW ArcList ArrowY BlankCircle
  EndSimulate HTransPlace MainMenu MainW Offset
  PWindowRegion PlaceCount PlaceHTrans PlaceLabels
  PlaceList PlaceVTrans PromptW Safe SequenceRegion
  SequenceW Status TokenList TokenSize TotalTokenCount
  TransCount TransLabels TransList TreeMarkings TreeRegion
  TreeW VTransPlace WindowRegion
Methods
  CalculatePoints DrawArc DrawArrowForTrans
  DrawFromTransition DrawToTransition RemoveSelf
  AreYouClicked BoundAPlace CheckCurrentPath CheckParent
  CheckTermination CompareObjs CreateSequence DoAnalysis
  DoInspect DoSimulation DoTree DoesRegionOverlap FindObj
  Fire FlushTokens GenerateTree GetEnabledTrans
  MessageNotUnderstood MethodNotFound ProcessArc
  ProcessCleanUp ProcessDelete ProcessHTrans ProcessPlace
  ProcessToken ProcessVTrans PutPrompt RemoveSequence
  SeeSequence SelectGrid SetSizes SetUpW StartSimulation
  SuperMethodNotFound UpdateSize WaitMouseDown
  WaitMouseDownGrid
```

Figure 4.3

User Exec -- PPDefault Window

#$Place

Supers

PetriNet

IVs

Bound Center InArcs InputList LabelRegion OutArcs OutputList PlaceName Radius Region TokenList Tokens Type

ArcInst BroomInst DeleteInst HTransInst PlaceInst Sequence SimulateInst SizeFlag TList TokenInst VTransInst

CVs

EraseCircle Radius

AnalysisRegion AnalysisW ArcList ArrowY BlankCircle Conservative EndSimulate HTransPlace MainMenu MainW Offset PWindowRegion PlaceCount PlaceHTrans PlaceLabels PlaceList PlaceVTrans PromptW Safe SequenceRegion SequenceW Status TokenList TokenSize TotalTokenCount TransCount TransLabels TransList TreeMarkings TreeRegion TreeW VTransPlace WindowRegion

Methods

AddInArc AddOutArc AddToInputFn AddToOutputFn AddToken CheckForBound CheckForToken CleanUp DecrementToken DeleteAToken DeleteFromInputList DeleteFromOutputList DoUpdate Draw DrawANewToken FlushTokens ProcessBound RedrawTokens RemoveSelf TokenCount

AreYouClicked BoundAPlace CheckCurrentPath CheckParent CheckTermination CompareObjs CreateSequence DoAnalysis DoInspect DoSimulation DoTree DoesRegionOverlap FindObj Fire GenerateTree GetEnabledTrans MessageNotUnderstood MethodNotFound ProcessArc ProcessCleanUp ProcessDelete ProcessHTrans ProcessPlace ProcessToken ProcessVTrans PutPrompt RemoveSequence SeeSequence SelectGrid SetSizes SetUpW StartSimulation SuperMethodNotFound UpdateSize WaitMouseDown WaitMouseDownGrid

Figure 4.4

User Exec -- PPDefault Window

SetUpW StartSimulation SuperMethodNotFound UpdateSize
WaitMouseDown WaitMouseDownGrid

#$Token

Supers

PetriNet

IVs

Center Region Type

ArcInst BroomInst DeleteInst HTransInst PlaceInst
Sequence SimulateInst SizeFlag TList TokenInst VTransInst

CVs

Radius

AnalysisRegion AnalysisW ArcList ArrowY BlankCircle
EndSimulate HTransPlace MainMenu MainW Offset
PWindowRegion PlaceCount PlaceHTrans PlaceLabels
PlaceList PlaceVTrans PromptW Safe SequenceRegion
SequenceW Status TokenList TokenSize TotalTokenCount
TransCount TransLabels TransList TreeMarkings TreeRegion
TreeW VTransPlace WindowRegion

Methods

CreateInPlace DoUpdate RemoveSelf

AreYouClicked BoundAPlace CheckCurrentPath CheckParent
CheckTermination CompareObjs CreateSequence DoAnalysis
DoInspect DoSimulation DoTree DoesRegionOverlap FindObj
Fire FlushTokens GenerateTree GetEnabledTrans
MessageNotUnderstood MethodNotFound ProcessArc
ProcessCleanUp ProcessDelete ProcessHTrans ProcessPlace
ProcessToken ProcessVTrans PutPrompt RemoveSequence
SeeSequence SelectGrid SetSizes SetUpW StartSimulation
SuperMethodNotFound UpdateSize WaitMouseDown
WaitMouseDownGrid

Figure 4.5

```
User Exec -- PPDefault Window
    #$Transition
Supers
   PetriNet
IVs
   FlashRegion LabelRegion TransName Type
   ArcInst BroomInst DeleteInst HTransInst PlaceInst
   Sequence SimulateInst SizeFlag TList TokenInst VTransInst
CVs
   HalfRegionLength Length RegionLength RegionWidth
   Thickness Width
   AnalysisRegion AnalysisW ArcList ArrowY BlankCircle
   EndSimulate HTransPlace MainMenu MainW Offset
   PWindowRegion PlaceCount PlaceHTrans PlaceLabels
   PlaceList PlaceVTrans PromptW Safe SequenceRegion
   SequenceW Status TokenList TokenSize TotalTokenCount
   TransCount TransLabels TransList TreeMarkings TreeRegion
   TreeW VTransPlace WindowRegion
Methods
   CheckForEnable CheckInputPlaces CheckNumArcs
   CleanUp DeleteFromInputList DeleteFromOutputList
   DoUpdate HighLightSelf ProcessEnabledTran
   RemoveSelf
   AreYouClicked BoundAPlace CheckCurrentPath CheckParent
   CheckTermination CompareObjs CreateSequence DoAnalysis
   DoInspect DoSimulation DoTree DoesRegionOverlap FindObj
   Fire FlushTokens GenerateTree GetEnabledTrans
   MessageNotUnderstood MethodNotFound ProcessArc
   ProcessCleanUp ProcessDelete ProcessHTrans ProcessPlace
   ProcessToken ProcessVTrans PutPrompt RemoveSequence
   SeeSequence SelectGrid SetSizes SetUpW StartSimulation
   SuperMethodNotFound UpdateSize WaitMouseDown
   WaitMouseDownGrid
```

Figure 4.6

User Exec -- PPDefault Window

#$HTransition

Supers

Transition

IVs

Center InArcs InputList OutArcs OutputList
Region Type
ArcInst BroomInst DeleteInst FlashRegion HTransInst
LabelRegion PlaceInst Sequence SimulateInst SizeFlag
TList TokenInst TransName VTransInst

CVs

AnalysisRegion AnalysisW ArcList ArrowY BlankCircle
Conservative EndSimulate HTransPlace HalfRegionLength
Length MainMenu MainW Offset PWindowRegion PlaceCount
PlaceHTrans PlaceLabels PlaceList PlaceVTrans PromptW
RegionLength RegionWidth Safe SequenceRegion SequenceW
Status Thickness TokenList TokenSize TotalTokenCount
TransCount TransLabels TransList TreeMarkings
TreeRegion TreeW VTransPlace Width WindowRegion

Methods

AddInArc AddOutArc AddToInputFn AddToOutputFn
Draw
AreYouClicked BoundAPlace CheckCurrentPath
CheckForEnable CheckInputPlaces CheckNumArcs
CheckParent CheckTermination CleanUp CompareObjs
CreateSequence DeleteFromInputList
DeleteFromOutputList DoAnalysis DoInspect DoSimulation
DoTree DoUpdate DoesRegionOverlap FindObj Fire
FlushTokens GenerateTree GetEnabledTrans HighLightSelf
MessageNotUnderstood MethodNotFound ProcessArc
ProcessCleanUp ProcessDelete ProcessEnabledTran
ProcessHTrans ProcessPlace ProcessToken ProcessVTrans
PutPrompt RemoveSelf RemoveSequence SeeSequence
SelectGrid SetSizes SetUpW StartSimulation
SuperMethodNotFound UpdateSize WaitMouseDown
WaitMouseDownGrid

Figure 4.7

User Exec -- PPDefault Window

#$VTransition

Supers

Transition

IVs

Center InArcs InputList OutArcs OutputList Region Type
ArcInst BroomInst DeleteInst FlashRegion HTransInst LabelRegion PlaceInst Sequence SimulateInst SizeFlag TList TokenInst TransName VTransInst

CVs

AnalysisRegion AnalysisW ArcList ArrowY BlankCircle Conservative EndSimulate HTransPlace HalfRegionLength Length MainMenu MainW Offset PWindowRegion PlaceCount PlaceHTrans PlaceLabels PlaceList PlaceVTrans PromptW RegionLength RegionWidth Safe SequenceRegion SequenceW Status Thickness TokenList TokenSize TotalTokenCount TransCount TransLabels TransList TreeMarkings TreeRegion TreeW VTransPlace Width WindowRegion

Methods

AddInArc AddOutArc AddToInputFn AddToOutputFn Draw
AreYouClicked BoundAPlace CheckCurrentPath CheckForEnable CheckInputPlaces CheckNumArcs CheckParent CheckTermination CleanUp CompareObjs CreateSequence DeleteFromInputList DeleteFromOutputList DoAnalysis DoInspect DoSimulation DoTree DoUpdate DoesRegionOverlap FindObj Fire FlushTokens GenerateTree GetEnabledTrans HighLightSelf MessageNotUnderstood MethodNotFound ProcessArc ProcessCleanUp ProcessDelete ProcessEnabledTran ProcessHTrans ProcessPlace ProcessToken ProcessVTrans PutPrompt RemoveSelf RemoveSequence SeeSequence SelectGrid SetSizes SetUpW StartSimulation SuperMethodNotFound UpdateSize WaitMouseDown WaitMouseDownGrid

Figure 4.8

'EraseCircle'. Any instance of this class can use the class variable 'EraseCircle' to perform the desired operation; in this case, the opertation is to erase a circle on the screen that represents a place. Under the headings Methods are found the procedures representing the methods, which are functions in Interlisp. For example, 'AddToken' is the name of a function that implements the incrementation of token count for instances of Place. Other methods are 'AddInArc', "AddToInputFn', and 'CheckForBound'. Methods that are shown in bold face are declared by this class, while the other methods are inherited from its super class. Examples of methods that are inherited from PetriNet by this class are 'AreYouClicked' and 'BoundAPlace'.

4.2 Implementation of the reachability tree

A Petri net may have several different reachability trees depending on how the tree is implemented. Most implementations follow a breadth-first search approach to generate the tree. The implementation of the reachability tree in PETRISYS follows a depth-first search. A new marking is generated for each node from the root of the tree and continue on that path unless a terminating condition is satisfied. A terminating condition is either a limit imposed by the system on the length of the current path that is being searched or no firing condition exists from the current marking. Backtracking is performed by going to the next highest level where a new marking can be generated from the last enabled transition that has not been processed. Processing continues on this path until the terminating conditions are again statisfied for markings that are generated. This search process continues as long as firing conditions exist into the limits imposed by the system on the depth of the reachability tree.

CHAPTER 5
CONCLUSION

5.1 Directions for further work

PETRISYS is by no means a complete system. Work on the different subcomponents of this system can be substantially extended. A very useful feature that allows users to save Petri nets to different files should be implemented so that these nets can be retrieved for later use. Due to the complexity of some systems being modeled, it may not be feasible or even possible to draw all the nodes in a net at a single level. Therefore, a useful editing system should allow the user to substitute a node by a subnet that defines its refinement or to substitute a subnet by a single node to allow for more abstract modeling. This feature would allow the user to work at various hierarchical levels. The graphical editor should allow a user to move or reposition net objects while keeping the basic structure of the Petri net intact. Another important extension is to allow for bending arrows in the Petri net. This feature is not available in the current implementation of PETRISYS. At the present time, PETRISYS can only check for the syntactic correctness of a net. Semantic correctness of the net is the responsibility of the user. A good extension would be to supplement the syntax checker with a semantic checker. Since PETRISYS allows for more than one token in each place, it makes sense to have more than one arc between places and transitions. Another extension that should prove useful in the simulation phase of PETRISYS allows the user to select a particular transition from among a list of enabled transitions to fire.

5.2 The importance of this work

One of the major problems during the past years has been the lack of a sufficiently sophisticated graphical Petri net editor. Moreover, the currently existing analysis tools are not consistent in the sense that they run on different machinery and use different input and output formats. Many future designers of Petri net analysis tools will likely build their own package on top of one of the existing high-quality graphical editors. This trend would make Petri net packages more compatible with each other. The motivation for this is that a high quality editor takes approximately 10 man-years to develop. As a result, three research institutions, GMD Bonn, Zaragosa University, and Aarhus University, have made plans to build their future Petri net packages upon the Macintosh[6] version of Design. Design [Design86] is a good graphical editor that currently runs on the Apple Computer's Macintoshes. The designer feels that this trend of standardization will likely continue in the near future. PETRISYS could serve as a starting point for a productive graphical editor that utilizes the power of workstations.

5.3 Concluding Remarks

PETRISYS is more than a representation system because analysis procedures are also provided. PETRISYS offers the user a graphical environment to edit, simulate, and analyze Petri nets. Integrity and consistency checks are performed by the system so that users can concentrate on the more important tasks of modeling and analysis. This system was implemented in an object-oriented approach on a Xerox 1186 A.I. workstation. By implementing PETRISYS on a graphical workstation, the user is offered both the precision and accuracy of net drawings as well as the speed and power of interactive presentations.

6Macintosh is a trademark of Apple Computer, Inc.

Furthermore, the user is freed from details of the underlying theory governing the system that is being modeled. With the recent interest in Petri net tools, PETRISYS offers another dimension into the modeling of Petri nets.

Bibliography

[Agerwala73] T. Agerwala, and M. Flynn, "Comments on Capabilities, Limitations and 'Correctness' of Petri nets," Hopkins Computer Research Report Number 26, John Hopkins University, Baltimore, Maryland, July 1973, 58 pages; Also Proceedings of the First Annual Symposium on Computer Architecture, New York: ACM, 1973, pp81-86.

[Baer73] Baer, J. L., "A survey of some theoretical aspects of multiprocessing," Computing Surveys, Volume 5, Number 1, March 1973, pp31-80.

[Beau83] M. Beaudouin-Lafon, "Petripote: A graphic system for Petri net design and simulation," Proceedings of the 4th European Workshop on Applications and Theory of Petri nets, Toulouse, France 1983, pp20-30.

[Bobrow83] Daniel G. Bobrow, and Mark Stefik, "The LOOPS Manual," Xerox Parc, December 1983.

[Denn70] J. B. Dennis, "Record of the project MAC Conference on Concurrent Systems and Parallel Computation," New York: ACM, June 1970, 199 pages.

[Design86] Design Users guide, Meta Software Corporation, Cambridge, Massachusetts, 1986.

[Feld86] F. Fredbrugge and K. Jensen, "Petri net tool overview - 1986," Lecture notes in Computer Science, number 255, Advances in Petri nets 1986, Part II, Proceedings of an Advanced Course, Bad Honnef, West Germany, September 1986, Springer-Verlag.

[Genrich83] H.J. Genrich and R.M. Shapiro, "A diagram editor for line drawings with inscriptions," A. Pagnoni and G. Rozenberg (eds.), Applications and Theory of Petri nets, Informatik-Fachberichte 66, Springer-Verlag 1983, pp112-131.

[Holt68] Holt A. W. et al, "Final report of the information system theory project", T.R. RADC-TR-68-305, Rome Air Development Center, Griffiss Air Force Base, N.Y., Sept. 1968.

[Karp69] R. M. Karp and R. E. Miller, "Parallel program schemata," Journal of Computer and Systems Science 3, 4, May 1969, pp167-195; Also IEEE Conference Record of the 1967 Eighth Annual Symposium on Switching and Automata Theory, New York: IEEE, October 1967, pp55-61.

[Mont83] B. Montel et al, "Ovide, A software package for the validation of systems represented by Petri net based models," Proceedings of the 4th European Workshop on Applications and Theory of Petri nets, Toulouse, France 1983, pp292-308.

[Pete77] J. Peterson, "Petri Nets", Computing Surveys, Volume 9, Number 3, September 1977.

[Petri62] C. Petri, "Kommunikation mit Automaten," Ph.D. dissertation, University of Bonn, Bonn, West Germany, 1962, Also M.I.T. Memorandum MAC-M-212, Project MAC, M.I.T., Cambridge, Massachusetts.

[Xerox83] Xerox Artificial Intelligence Systems, Xerox Palo Alto Research Center, 1983, Interlisp-D Reference Manuals;

Volume 1: Languages,

Volume 2: Environments,

Volume 3: Input/Output.

Appendix

Source Code Listing

```
(* Copyright (c) 1988 by Xerox Corporation. All rights reserved.)

(PRETTYCOMPRINT PLACEFILECOMS)

(RPAQQ PLACEFILECOMS ((* File created by )
        (CLASSES Place)
        (METHODS Place.AddInArc Place.AddOutArc Place.AddToInputFn Place.AddToOutputFn Place.AddToken
                 Place.CheckForBound Place.CheckForToken Place.DecrementToken Place.DeleteAToken
                 Place.DeleteFromInputList Place.DeleteFromOutputList Place.DoUpdate Place.Draw
                 Place.DrawANewToken Place.FindCorrectLabel Place.FlushTokens Place.ProcessBound
                 Place.RedrawTokens Place.RemoveSelf Place.TokenCount)
        (FNS)
        (VARS)
        (INSTANCES)))


(* File created by )

(DEFCLASSES Place)
[DEFCLASS Place
   (MetaClass Class Edited:                                        (* edited: "15-Mar-88 09:06"))
   (Supers PetriNet)
   (ClassVariables (Radius 21 doc                                  (* CV added by))
                   (EraseCircle 21 doc                             (* CV added by)))
   (InstanceVariables (Center (0.0)
                              doc                                  (* IV added by)
                              )
                      (Region (0.0)
                              doc                                  (* IV added by)
                              )
                      (Radius 18 doc                               (* IV added by))
                      (Tokens 0 doc                                (* IV added by))
                      (Type Place doc                              (* IV added by))
                      (InputList NIL doc                           (* IV added by))
                      (OutputList NIL doc                          (* IV added by))
                      (TokenList NIL doc                           (* IV added by))
                      (InArcs NIL doc                              (* IV added by))
                      (OutArcs NIL doc                             (* IV added by))
                      (PlaceName NIL doc                           (* IV added by TAN))
                      (LabelRegion NIL doc                         (* IV added by))
                      (Bound 10 doc                                (* IV addad by))))]
[METH Place  AddInArc (ArcInst)
      (* Adds an incoming arc instance into Place)]


[METH Place  AddOutArc (ArcInst)
      (* New method template)]


[METH Place  AddToInputFn (Transition)
      (* adds instance of transition into the input list for place)]


[METH Place  AddToOutputFn (Transition)
      (* Adds an instance of transition into the output list for Place)]


[METH Place  AddToken (Token)
```

```
        (* Increment token count in Place and add the token into token list)]

[METH Place  CheckForBound NIL
        (* New method template)]

[METH Place  ChackForToken NIL
        (* returns true if there is at least one token in the place)]

[METH Place  DecramentToken (token)
        (* update token count in Place and token list when a token has been deleted)]

[METH Place  DeleteAToken NIL
        (* When simulating, always delete the first token in the list)]

[METH Place  DeleteFromInputList (Trans)
        (* remove the Transition from input list of Place)]

[METH Place  DeleteFromOutputList (Trans)
        (* remove the Transition from output list of Place)]

[METH Place  DoUpdate (size)
        (* New method template)]

[METH Place  Draw (Center)
        (* Draws the Place instance and update global Place list)]

[METH Place  DrawANewToken NIL
        (* will request Token object to draw a new token)]

[METH Place  FindCorrectLabel NIL
        (* New method template)]

[METH Place  FlushTokens NIL
        (* New mathod template)]

[METH Place  ProcassBound NIL
        (* New method template)]

[METH Place  RedrawTokens NIL
        (* will radraw the tokens in a Place after an arc has been drawn)]

[METH Place  RamoveSelf NIL
        (* take care of inarcs, outarcs, inputlist, outputlist, and global variablas)]

[METH Place  TokanCount NIL
        (* New method template)]

(DEFINEQ
```

```
(Place.AddInArc
  [Method ((Place AddInArc)
           self ArcInst)                     (* edited: "13-Aug-87 10:05")
                                             (* Adds an incoming arc instance into Place)
          (_@
            :InArcs
            (CONS ArcInst (@ :InArcs])

(Place.AddOutArc
  [Method ((Place AddOutArc)
           self ArcInst)                     (* edited: "29-Jun-87 08:51")
                                             (* New method template)
          (_@
            :OutArcs
            (CONS ArcInst (@ :OutArcs])

(Place.AddToInputFn
  [Method ((Place AddToInputFn)
           self Transition)                  (* edited: "29-Jun-87 09:11")
                                             (* adds instance of transition into the input list for
                                             place)
          (_@
            :InputList
            (CONS Transition (@ :InputList])

(Place.AddToOutputFn
  [Method ((Place AddToOutputFn)
           self Transition)                  (* edited: "13-Aug-87 10:06")
                                             (* Adds an instance of transition into the output list
                                             for Place)
          (_@
            :OutputList
            (CONS Transition (@ :OutputList])

(Place.AddToken
  [Method ((Place AddToken)
           self Token)                       (* edited: "24-Nov-87 09:52")
                                             (* Increment token count in Place and add the token
                                             into token list)
                                             (* For Simulation)
          (_@
            :Tokens
            (ADD1 (@ :Tokens)))
          (if (LESSP (@ :Tokens)
                     5)
              then (_@
                      :TokenList
                      (CONS Token (@ :TokenList])

(Place.CheckForBound
  (Method ((Place CheckForBound)
           self)                             (* edited: "15-Sep-87 10:11")
                                             (* New method template)
          (if (LESSP (@ :Tokens)
                     (@ :Bound))
              then T
            else NIL)))
```

```
(Place.CheckForToken
  (Method ((Place CheckForToken)
           self)                                      (* edited: " 1-Sep-87 10:05")
                                                      (* returns true if there is at least one token in the
                                                       place)

         (COND
           ((GEO (@ :Tokens)
                 1)
            T)
           (T NIL))))

(Place.DecrementToken
  [Method ((Place DecrementToken)
           self token)                                (* edited: "13-Aug-87 10:07")
                                                      (* update token count in Place and token list when a
                                                       token has been deleted)

         (LET ((CurToken NIL)
               (List (@ :TokenList)))
              (_@
                :TokenList NIL)
              (while (NOT (NULL List))
                 do (SETQ CurToken (CAR List))
                    [if (NEQ CurToken token)
                        then (_@
                               :TokenList
                               (CONS CurToken (@ :TokenList]
                    (SETQ List (CDR List)))
              (_@
                :Tokens
                (SUB1 (@ :Tokens])

(Place.DeleteAToken
  [Method ((Place DeleteAToken)
           self)                                      (* edited: "22-Mar-88 10:13")
                                                      (* When simulating, always delete the first token in
                                                       the list)

         [if (LESSP (@ :Tokens)
                    5)
             then [LET ((FirstToken NIL)
                        (CurToken NIL)
                        (List (@ :TokenList))
                        (GList (@ ::TokenList)))
                       (SETQ FirstToken (CAR List))   (* blank first token in local token list)
                       (FILLCIRCLE (CAR (@ FirstToken :Center))
                                   (CDR (@ FirstToken :Center))
                                   (@ ::TokenSize)
                                   0
                                   (@ ::MainW))
                       (SETQ List (CDR List))
                       (_@
                         :TokenList NIL)              (* remove that token from local list)
                       (while (NOT (NULL List))
                          do (SETQ CurToken (CAR List))
                             (_@
                               :TokenList
                               (CONS CurToken (@ :TokenList)))
                             (SETQ List (CDR List)))
                       (_@
                         ::TokenList NIL)             (* remove same token from global token list)
                       (while (NOT (NULL GList))
                          do (SETQ CurToken (CAR GList))
                             [if (NEQ CurToken FirstToken)
                                 then (_@
                                        ::TokenList
```

```
                                            (CONS CurToken (@ ::TokenList]
                                   (SETQ GList (CDR GList)
        else (if (EQ (@ :Tokens)
                     5)
                 then (FILLCIRCLE (CAR (@ :Center))
                                  (CDR (@ :Center))
                                  (@ ::BlankCircle)
                                  0
                                  (@ ::MainW))
                      (FILLCIRCLE (DIFFERENCE (CAR (@ :Center))
                                              (@ ::Offset))
                                  (PLUS (CDR (@ :Center))
                                        (@ ::Offset))
                                  (@ ::TokenSize)
                                  TokenTexture
                                  (@ ::MainW))
                      (FILLCIRCLE (PLUS (CAR (@ :Center))
                                        (@ ::Offset))
                                  (PLUS (CDR (@ :Center))
                                        (@ ::Offset))
                                  (@ ::TokenSize)
                                  TokenTexture
                                  (@ ::MainW))
                      (FILLCIRCLE (DIFFERENCE (CAR (@ :Center))
                                              (@ ::Offset))
                                  (DIFFERENCE (CDR (@ :Center))
                                              (@ ::Offset))
                                  (@ ::TokenSize)
                                  TokenTexture
                                  (@ ::MainW))
                      (FILLCIRCLE (PLUS (CAR (@ :Center))
                                        (@ ::Offset))
                                  (DIFFERENCE (CDR (@ :Center))
                                              (@ ::Offset))
                                  (@ ::TokenSize)
                                  TokenTexture
                                  (@ ::MainW))
               else (FILLCIRCLE (CAR (@ :Center))
                                (CDR (@ :Center))
                                (@ ::BlankCircle)
                                0
                                (@ ::MainW))
                    (MOVETO (CAR (@ :Center))
                            (CDR (@ :Center))
                            (@ ::MainW))
                    (PRIN1 (SUB1 (@ :Tokens))
                           (@ ::MainW]
        (_@
          :Tokens
          (SUB1 (@ :Tokens)))
        (_@
          ::TotalTokenCount
          (SUB1 (@ ::TotalTokenCount])

(Place.DeleteFromInputList
  [Method ((Place DaleteFromInputList)
          self Trans)                                  (* edited: " 5-Jan-88 10:30")
                                                       (* remove the Transition from input list of Placa)
        [LET ((Obj NIL)
              (List (@ :InputList)))
             (_@
               :InputList NIL)
             (while (NOT (NULL List)) do ((SETQ Obj (CAR List))
                                          [if (NEQ Obj Trans)
```

```
                                                               then (_@
                                                                       :InputList
                                                                       (CONS Obj (@ :InputList]
                                                            (SETU List (COR List]
          (if (ANO (NULL (@ :InputList))
                   (NULL (@ :OutputList)))
              then (_ self RemoveSelf)
            else (ORAWCIRCLE (CAR (@ :Center))
                             (CDR (@ :Center))
                             (@ ::Radius)
                             NIL NIL (@ ::MainW])

(Place.DeleteFromOutputList
  [Method ((Place OeleteFromOutputList)
           self Trans)                                          (* edited: " 5-Jan-88 10:30")
                                                                (* remove the Transition from output list of Place)
          [LET ((Obj NIL)
                (List (@ :OutputList)))
               (_@
                 :OutputList NIL)
               (while (NOT (NULL List)) do ((SETO Obj (CAR List))
                                                            [if (NEQ Obj Trans)
                                                                then (_@
                                                                       :OutputList
                                                                       (CONS Obj (@ :OutputList]
                                                            (SETU List (COR List]
          (if (ANO (NULL (@ :InputList))
                   (NULL (@ :OutputList)))
              then (_ self RemoveSelf)
            else (DRAWCIRCLE (CAR (@ :Center))
                             (COR (@ :Center))
                             (@ ::Radius)
                             NIL NIL (@ ::MainW])

(Place.DoUpdate
  [Method ((Place DoUpdate)
           self size)                                           (* edited: " 6-Nov-87 D8:54")
                                                                (* New method template)
          (CONO
            ((EQ size (OUOTE Big))
              (_@
                ::Radius 21)
              (_@
                ::BlankCircle 19)
              (_@
                ::EraseCircla 21))
            ((EO size (OUOTE Medium))
              (_@
                ::Radius 18)
              (_@
                ::BlankCircla 16)
              (_@
                ::EraseCircle 18))
            ((EQ size (QUOTE Small))
              (_@
                ::Radius 15)
              (_@
                ::BlankCircle 13)
              (_@
                ::EraseCircle 15])
```

```
(Place.Draw
  [Method ((Place Draw)
           self Center)                                   (* edited: "25-Mar-88 14:12")
                                                          (* Draws the Place instance and update global Place
                                                             list)
          (_@
            :Center Center)
          (_@
            :Region
            (LIST (DIFFERENCE (DIFFERENCE (CAR (@ :Center))
                                          (@ :Radius))
                              5)
                  (DIFFERENCE (DIFFERENCE (CDR (@ :Center))
                                          (@ :Radius))
                              5)
                  (PLUS (TIMES (@ :Radius)
                               2)
                        10)
                  (PLUS (TIMES (@ :Radius)
                               2)
                        10)))
          (LET ((Label NIL))
               (COND
                 ((NOT (_ self DoesRegionOverlap (@ :Region)))
                   (DRAWCIRCLE (CAR (@ :Center))
                               (CDR (@ :Center))
                               (@ ::Radius)
                               NIL NIL (@ ::MainW))
                   (_@
                     ::PlaceCount
                     (ADD1 (@ ::PlaceCount)))
                   [if (NULL (@ ::PlaceDeleteFlag))
                       then (SETQ Label (PACK* (QUOTE p)
                                               (@ ::PlaceCount)))
                     else (if (NEQ (@ ::PlaceCount)
                                   0)
                              then (SETQ Label (PACK* (QUOTE p)
                                                      (_ self FindCorrectLabel]
                   (MOVETO (PLUS (PLUS (CAR (@ :Center))
                                       (@ :Radius))
                                 4)
                           (DIFFERENCE (CDR (@ :Center))
                                       (@ :Radius))
                           (@ ::MainW))
                   (PRIN1 Label (@ ::MainW))
                   (_@
                     :LabelRegion
                     (LIST (PLUS (CAR (@ :Center))
                                 (@ :Radius))
                           (DIFFERENCE (CDR (@ :Center))
                                       20)
                           24 10))
                   (_@
                     ::SizeFlag T)
                   (_@
                     ::PlaceLabels
                     (CONS Label (@ ::PlaceLabels)))
                   (_@
                     :PlaceName Label)
                   (_@
                     ::PlaceList
                     (CONS self (@ ::PlaceList)))
                   (CLEARW (@ ::PromptW)))
                 (T (PRINTBELLS)
```

```
                    (_ self PutPrompt "Put the Object somewhere else"])

(Place.DrawANewToken
  (Method ((Place DrawANewToken)
           self)                                             (* edited: "24-Nov-87 10:00")
                                                             (* will request Token object to draw a new token)
          (if (LESSP (@ :Tokens)
                     (@ :Bound))
              then (_ (_ ($ Token)
                            New)
                         CreateInPlace self (QUOTE Simulation))
            else (_ self PutPrompt "Bound reached for this place"))))

(Place.FindCorrectLabel
  (Method ((Place FindCorrectLabel)
           self)                                             (* edited: "22-Mar-88 20:09")
                                                             (* New method template)
          (LET ((List (@ ::PlaceList))
                (Done NIL)
                (Label NIL)
                (Count 1)
                (Found NIL)
                (CurPlace NIL))
               (SETQ CurPlace (CAR List))
               (while (NOT Found)
                  do (SETQ Done NIL)
                     (while (AND (NOT Done)
                                 (NOT (NULL List)))
                        do (SETQ List (CDR List))
                           (SETQ Label (UNPACK (@ CurPlace :PlaceName)))
                           (if (NOT (NULL (CDDR Label)))
                               then (SETQ Number (PACK* (CADR Label)
                                                        (CADDR Label)))
                             else (SETQ Number (CADR Label)))
                           (if (EQP Count Number)
                               then (SETQ Done T)
                                    (SETQ Count (ADD1 Count))
                                    (SETQ List (@ ::PlaceList))
                                    (SETQ CurPlace (CAR List))
                             else (SETQ CurPlace (CAR List]
                     (if (AND (NULL List)
                              (NOT Done))
                         then (SETQ Found T)))
          Count)))

(Place.FlushTokens
  (Method ((Place FlushTokens)
           self)                                             (* edited: "28-Oct-87 10:19")
                                                             (* New method template)
          (FILLCIRCLE (CAR (@ CurPlace :Center))
                      (CDR (@ CurPlace :Center))
                      (@ ::BlankCircle)
                      0
                      (@ ::MainW))
          (_@
            :TokenList NIL)
          (_@
            :Tokens 0)))

(Placa.ProcassBound
  (Method ((Place ProcessBound)
           self)                                             (* aditad: " 6-Jan-88 09:27")
                                                             (* New method template)
          (CLEARW (@ ::PromptW))
```

```
          [_@
            :Bound
            (PACK* (PROMPTFORWORD "Give a bound for this place > " NIL NIL (@ ::PromptW)
                                  NIL
                                  (OUOTE TTY]
          (_ self PutPrompt "Place has been bounded")))

(Place.RedrawTokens
  [Method ((Place RedrawTokens)
           self)                                          (* edited: "16-Mar-88 10:27")
                                                          (* Will redraw the tokens in a Place after an arc has
                                                          been drawn)
          (FILLCIRCLE (CAR (@ :Center))
                      (CDR (@ :Center))
                      (@ ::BlankCircle)
                      0
                      (@ ::MainW))
          (if (LEQ (@ :Tokens)
                   4)
              then [LET ((CurToken NIL)
                         (List (@ :TokenList)))
                        (while (NOT (NULL List))
                           do (SETQ CurToken (CAR List))
                              (FILLCIRCLE (CAR (@ CurToken :Center))
                                          (CDR (@ CurToken :Center))
                                          (@ ::TokenSize)
                                          TokenTexture
                                          (@ ::MainW))
                              (SETQ List (CDR List]
            else (MOVETO (CAR (@ :Center))
                         (CDR (@ :Center))
                         (@ ::MainW))
                 (PRIN1 (@ :Tokens)
                        (@ ::MainW))
                 (_@
                   ::TotalTokenCount
                   (ADD1 (@ ::TotalTokenCount])

(Place.RemoveSelf
  (Method ((Place RemoveSelf)
           self)                                          (* edited: "25-Mar-88 15:14")
                                                          (* take care of inarcs, outarcs, inputlist,
                                                          outputlist, and global variables)
          (_@
            ::PlaceDeleteFlag T)
          (FILLCIRCLE (CAR (@ :Center))
                      (CDR (@ :Center))
                      (@ ::EraseCircle)
                      0
                      (@ ::MainW))
          (SHADEGRIDBOX 0 0 0 NIL (@ :LabelRegion)
                        NIL
                        (@ ::MainW))
          [LET ((Obj NIL)
                (List (@ ::PlaceList)))
               (_@
                 ::PlaceList NIL)
               (while (NOT (NULL List)) do ((SETQ Obj (CAR List))
                                            [if (NEQ Obj self)
                                                then (_@
                                                       ::PlaceList
                                                       (CONS Obj (@ ::PlaceList]
                                            (SETQ List (CDR List]
          [if (NOT (NULL (@ :InArcs)))
```

```
          then (LET ((List (@ :InArcs))
                     (CurArc NIL))
                    (while (NOT (NULL List))
                       do (SETQ CurArc (CAR List))
                          (_ CurArc RemoveSelf)
                          (SETQ List (CDR List]
      (_@
        :InArcs NIL)
      [if (NOT (NULL (@ :OutArcs)))
          then (LET ((List (@ :OutArcs))
                     (CurArc NIL))
                    (while (NOT (NULL List))
                       do (SETQ CurArr (CAR List))
                          (_ CurArc RemoveSelf)
                          (SETQ List (CDR List]
      (_@
        :OutArcs NIL)
      [if (NOT (NULL (@ :InputList)))
          then (LET ((List (@ :InputList))
                     (Obj NIL))
                    (while (NOT (NULL List))
                       do (SETQ Obj (CAR List))
                          (_ Obj DeleteFromOutputList self)
                          (SETQ List (CDR List]
      (_@
        :InputList NIL]
      [if (NOT (NULL (@ :OutputList)))
          then (LET ((List (@ :OutputList))
                     (Obj NIL))
                    (while (NOT (NULL List))
                       do (SETQ Obj (CAR List))
                          (_ Obj DeleteFromInputList self)
                          (SETQ List (CDR List]
      (_@
        :OutputList NIL)
      [LET ((Label (@ :PlaceName))
            (List (@ ::PlaceLabels)))
           (_@
             ::PlacaLabels NIL)
           (while (NOT (NULL List)) do ([if (NEQ Lebel (CAR List))
                                            then (_@
                                                   ::PlaceLabels
                                                   (CONS (CAR List)
                                                         (@ ::PleceLabels]
                                        (SETQ List (CDR List]
      (LET ((Found NIL)
            (LocelToken NIL)
            (GlobelToken NIL)
            (LocalList (@ :TokenList))
            (GlobalList (@ ::TokenList)))
           (while (NOT (NULL LocalList))
              do (SETQ LocelToken (CAR LocelList))
                 (while (AND (NOT Found)
                             (NOT (NULL GlobalList)))
                    do (if (EQUAL LocelTokan (CAR GlobalList))
                           then (REMOVE (QUOTE LocalToken)
                                        (QUOTE GlobalList))
                                (_@
                                  ::TotelTokanCount
                                  (SUB1 (@ ::TotelTokenCount)))
                                (SETQ Found T))
                       (SETQ GlobelList (CDR GlobelList)))
                 (SETQ LocelList (CDR LocalList))
                 (SETQ Found NIL))
```

```
                (_@
                   ::TokenList GlobalList))
          (_@
             ::PlaceCount
             (SUB1 (@ ::PlaceCount)))
          (_@
             :Tokens 0)
          (_@
             :Region
             (QUOTE (0 0 0 0)))
          (_@
             :Center
             (QUOTE (0 0)))
          (_@
             :TokenList NIL)))

(Place.TokenCount
  (Method ((Place TokenCount)
           self)                                              (* edited: " 7-Mar-88 08:03")
                                                              (* New method template)
          (@ :Tokens)))
(PUTPROPS PLACEFILE COPYRIGHT ("Xerox Corporation" 1988))
(DECLARE: DONTCOPY
  (FILEMAP (NIL (3886 21407 (Place.AddInArc 3896 . 4197) (Place.AddOutArc 4199 . 4483) (
Place.AddToInputFn 4485 . 4836) (Place.AddToOutputFn 4838 . 5197) (Place.AddToken 5199 . 5740) (
Place.CheckForBound 5742 . 6065) (Place.CheckForToken 6067 . 6430) (Place.DecrementToken 6432 . 7121)
(Place.DeleteAToken 7123 . 10040) (Place.DeleteFromInputList 10042 . 10875) (
Place.DeleteFromOutputList 10877 . 11717) (Place.DoUpdate 11719 . 12336) (Place.Draw 12338 . 14245) (
Place.DrawANewToken 14247 . 14735) (Place.FindCorrectLabel 14737 . 15935) (Place.FlushTokens 15937 .
16346) (Place.ProcessBound 16348 . 16812) (Place.RedrawTokens 16814 . 17829) (Place.RemoveSelf 17831
 . 21158) (Place.TokenCount 21160 . 21405)))))
STOP
```

{DSK}<LISPFILES>PLACEFILE.;1 6-Apr-88 09:13:39
-- Listed on 6-Apr-88 09:14:46 --

FUNCTION INDEX

```
(* Copyright (c) 1988 by Xerox Corporation. All rights reserved.)

(PRETTYCOMPRINT TOKENFILECOMS)

(RPAQQ TOKENFILECOMS ((* File created by )
                      (CLASSES Token)
                      (METHODS Token.CreateInPlace Token.DoUpdate Token.RemoveSelf)
                      (FNS)
                      (VARS)
                      (INSTANCES)))


(* File created by )

(DEFCLASSES Token)
[DEFCLASS Token
   (MetaClass Class Edited:                                  (* edited: " 4-Nov-87 14:36"))
   (Supers PetriNet)
   (ClassVariables (Radius 1 doc                             (* CV added by)))
   (InstanceVariables (Center (0.0)
                              doc                            (* IV added by)
                              )
                      (Type Token doc                        (* IV added by))
                      (Region (0.0)
                              doc                            (* IV added by)
                              ))]

[METH Token  CreateInPlace (PlaceInst Type)
      (* New method template)]


[METH Token  DoUpdate (size)
      (* New method template)]


[METH Token  RemoveSelf (pt)
      (* Remove token during deletion and update global token list as well as total token count)]


(DEFINEQ


(Token.CreateInPlace
  [Method ((Token CreateInPlace)
           self PlaceInst Type)                              (* edited: "22-Mar-88 10:14")
                                                             (* New method template)
          [LET ((Center (@ PlaceInst :Center)))
               (if (EQ (@ PlaceInst :Tokens)
                       0)
                   then (FILLCIRCLE (DIFFERENCE (CAR Center)
                                                (@ ::Offset))
                                    (PLUS (CDR Center)
                                          (@ ::Offset))
                                    (@ ::TokenSize)
                                    TokenTexture
                                    (@ ::MainW))
                        [_@
                          :Center
```

```
                    (CONS (DIFFERENCE (CAR Center)
                                      (@ ::Offset))
                          (PLUS (CDR Center)
                                (@ ::Offset]
        else (if (GEQ (@ PlaceInst :Tokens)
                      1)
                 then (FILLCIRCLE (CAR Center)
                                  (CDR Center)
                                  (@ ::BlankCircle)
                                  0
                                  (@ ::MainW))
                      (COND
                        [(EQ (@ PlaceInst :Tokens)
                             1)
                          (FILLCIRCLE (DIFFERENCE (CAR Center)
                                                  (@ ::Offset))
                                      (PLUS (CDR Center)
                                            (@ ::Offset))
                                      (@ ::TokenSize)
                                      TokenTexture
                                      (@ ::MainW))
                          (FILLCIRCLE (PLUS (CAR Center)
                                            (@ ::Offset))
                                      (PLUS (CDR Center)
                                            (@ ::Offset))
                                      (@ ::TokenSize)
                                      TokenTexture
                                      (@ ::MainW))
                          (_@
                            :Center
                            (CONS (PLUS (CAR Center)
                                        (@ ::Offset))
                                  (PLUS (CDR Center)
                                        (@ ::Offset]
                        [(EQ (@ PlaceInst :Tokens)
                             2)
                          (FILLCIRCLE (DIFFERENCE (CAR Center)
                                                  (@ ::Offset))
                                      (PLUS (CDR Center)
                                            (@ ::Offset))
                                      (@ ::TokenSize)
                                      TokenTexture
                                      (@ ::MainW))
                          (FILLCIRCLE (PLUS (CAR Center)
                                            (@ ::Offset))
                                      (PLUS (CDR Center)
                                            (@ ::Offset))
                                      (@ ::TokenSize)
                                      TokenTexture
                                      (@ ::MainW))
                          (FILLCIRCLE (DIFFERENCE (CAR Center)
                                                  (@ ::Offset))
                                      (DIFFERENCE (CDR Center)
                                                  (@ ::Offset))
                                      (@ ::TokenSize)
                                      TokenTexture
                                      (@ ::MainW))
                          (_@
                            :Center
                            (CONS (DIFFERENCE (CAR Center)
                                              (@ ::Offset))
                                  (DIFFERENCE (CDR Center)
                                              (@ ::Offset]
                        [(EQ (@ PlaceInst :Tokens)
```

```
                                            3)
                                      (FILLCIRCLE (DIFFERENCE (CAR Center)
                                                                          (@ ::Offset))
                                                  (PLUS (CDR Center)
                                                              (@ ::Offset))
                                                  (@ ::TokenSize)
                                                  TokenTexture
                                                  (@ ::MainW))
                                      (FILLCIRCLE (PLUS (CAR Center)
                                                              (@ ::Offset))
                                                  (PLUS (CDR Center)
                                                              (@ ::Offset))
                                                  (@ ::TokenSize)
                                                  TokenTexture
                                                  (@ ::MainW))
                                      (FILLCIRCLE (DIFFERENCE (CAR Center)
                                                                          (@ ::Offset))
                                                  (DIFFERENCE (CDR Center)
                                                                          (@ ::Offset))
                                                  (@ ::TokenSize)
                                                  TokenTexture
                                                  (@ ::MainW))
                                      (FILLCIRCLE (PLUS (CAR Center)
                                                              (@ ::Offset))
                                                  (DIFFERENCE (CDR Center)
                                                                          (@ ::Offset))
                                                  (@ ::TokenSize)
                                                  TokenTexture
                                                  (@ ::MainW))
                                      (_@
                                        :Center
                                        (CONS (PLUS (CAR Center)
                                                          (@ ::Offset))
                                                    (DIFFERENCE (CDR Center)
                                                                      (@ ::Offset]
                                    (T (FILLCIRCLE (CAR Center)
                                                   (CDR Center)
                                                   (@ ::BlankCircle)
                                                   0
                                                   (@ ::MainW))
                                       (MOVETO (DIFFERENCE (CAR Center)
                                                                 3)
                                               (CDR Center)
                                               (@ ::MainW))
                                       (PRIN1 (ADD1 (@ PlaceInst :Tokens))
                                               (@ ::MainW)
              (_ PlaceInst AddToken self)
              (_@
                ::TotalTokenCount
                (ADD1 (@ ::TotalTokenCount)))
              (_@
                ::TokenList
                (CONS self (@ ::TokenList])

(Token.DoUpdate
  [Method ((Token DoUpdate)
            self size)                                          (* edited: " 4-Nov-87 14:50")
                                                                (* New method template)
            (COND
              ((EQ size (QUOTE Big))
                (_@
                  ::Offset 8)
                (_@
                  ::TokenSize 5))
```

```
              ((EQ size (QUOTE Medium))
                (_@
                  ::Offset 7)
                (_@
                  ::TokenSize 4))
              ((EQ size (QUOTE Small))
                (_@
                  ::Offset 6)
                (_@
                  ::TokenSize 3])

(Token.RemoveSelf
  [Method ((Token RemoveSelf)
           self pt)                                         (* edited: "28-Oct-87 10:24")
                                                            (* Remove token during deletion and update global
                                                             token list as well as total token count)
           (FILLCIRCLE (CAR (@ :Center))
                       (CDR (@ :Center))
                       (@ ::TokenSize)
                       0
                       (@ ::MainW))
           (_@
             ::TotalTokenCount
             (SUB1 (@ ::TotalTokenCount)))
           (LET ((CurToken NIL)
                 (Found NIL)
                 (List (@ ::TokenList)))
                (_@
                  ::TokenList NIL)
                (while (NOT (NULL List))
                   do (SETQ CurToken (CAR List))
                        [if (NEQ CurToken self)
                            then (_@
                                   ::TokenList
                                   (CONS CurToken (@ ::TokenList]
                        (SETQ List (CDR List)))
                (SETQ List (@ ::PlaceList))
                (while (AND (NOT Found)
                            (NOT (NULL List)))
                   do (SETQ Found (INSIDEP (@ (CAR List)
                                              :Region)
                                           (CAR pt)
                                           (CDR pt)))
                        (if (NOT Found)
                            then (SETQ List (CDR List))
                          else (_ (CAR List)
                                  DecrementToken self])))
(PUTPROPS TOKENFILE COPYRIGHT ("Xerox Corporation" 1988))
(DECLARE: DONTCOPY
  (FILEMAP (NIL (1206 6975 (Token.CreateInPlece 1216 . 5237) (Token.DoUpdate 5239 . 5751) (
Token.RemoveSelf 5753 . 6973)))))
STOP
```

{OSK}<LISPFILES>TOKENFILE.;1

{DSK}<LISPFILES>TOKENFILE.;1 6-Apr-88 09:01:42
-- Listed on 6-Apr-88 09:02:47 --

FUNCTION INDEX

```
(* Copyright (c) 1988 by Xerox Corporation. All rights reserved.)

(PRETTYCOMPRINT TRANSFILECOMS)

(RPAQQ TRANSFILECOMS ((* File created by )
        (CLASSES HTransition Transition VTransition)
        (METHODS HTransition.AddInArc HTransition.AddOutArc HTransition.AddToInputFn
                 HTransition.AddToOutputFn HTransition.Draw Transition.CheckForEnable
                 Transition.CheckInputPlaces Transition.CheckNumArcs Transition.DeleteFromInputList
                 Transition.DeleteFromOutputList Transition.DnUpdate Transition.FindCorrectLabel
                 Transition.HighLightSelf Transition ProcessEnabledTran Transition.RemoveSelf
                 VTransition.AddInArc VTransition.AddOutArc VTransition.AddToInputFn
                 VTransition.AddToOutputFn VTransition.Draw)
        (FNS)
        (VARS)
        (INSTANCES)))


(* File created by )

(DEFCLASSES HTransition Transition VTransition)
[DEFCLASS HTransition
   (MetaClass Class Edited:                                    (* edited: " 4-May-87 09:52"))
   (Supers Transition)
   (InstanceVariables (Center **DefaultValue doc               (* IV added by))
                      (Region **DefaultValue doc               (* IV added by))
                      (Type HTrans doc                         (* IV added by))
                      (InputList NIL doc                       (* IV added by))
                      (OutputList NIL doc                      (* IV added by))
                      (InArcs NIL doc                          (* IV added by))
                      (OutArcs NIL doc                         (* IV added by))))]

[DEFCLASS Transition
   (MetaClass Class Edited:                                    (* edited: "13-Mar-88 12:39"))
   (Supers PetriNet)
   (ClassVariables (Length 19 doc                              (* CV added by))
                   (Width 4 doc                                (* CV added by))
                   (RegionLength 42 doc                        (* CV added by))
                   (RegionWidth 8 doc                          (* CV added by))
                   (HalfRegionLength 22 doc                    (* CV added by))
                   (Thickness 4 doc                            (* CV added by)))
   (InstanceVariables (Type Transition doc                     (* IV added by))
                      (TransName NIL doc                       (* IV added by TAN))
                      (LabelRegion NIL doc                     (* IV added by))
                      (FlashRegion NIL doc                     (* IV added by))))]

[DEFCLASS VTransition
   (MetaClass Class Edited:                                    (* edited: " 4-May-87 09:38"))
   (Supers Transition)
   (InstanceVariables (Center **DefaultValue doc               (* IV added by))
                      (Region **DefaultValue doc               (* IV added by))
                      (Type VTrans doc                         (* IV added by))
                      (InputList NIL doc                       (* IV added by))
                      (OutputList NIL doc                      (* IV added by))
                      (InArcs NIL doc                          (* IV added by))
                      (OutArcs NIL doc                         (* IV added by))))]

[METH HTransition  AddInArc (ArcInst)
      (* add new arc drawn into InArcs instance)]
```

```
[METH HTransition  AddOutArc (ArcInst)
      (* adds new arc drawn into OutArcs instance)]

[METH HTransition  AddToInputFn (Place)
      (* Adds horizontal transition into the list of Place)]

[METH HTransition  AddtoOutputFn (Place)
      (* Adds horizontal transition into output list of Place)]

[METH HTransition  Draw (Center)
      (* Transition region has length 42 and width 11)]

[METH Transition  CheckForEnable NIL
      (* check each place coming into a transition to see if the transition is enabled)]

[METH Transition  CheckInputPlares NIL
      (* New method template)]

[METH Transition  CherkNumArcs NIL
      (* New method template)]

[METH Transition  DeleteFromInputList (Place)
      (* remove the Place instance from input list of transition)]

[METH Transition  DeleteFromOutputList (Place)
      (* remove the Place instance from output list of transition)]

[METH Transition  DoUpdate (size)
      (* New method template)]

[METH Transition  FindCorrectLabel NIL
      (* New mathod template)]

[METH Transition  HighLightSelf NIL
      (* New method templata)]

[METH Transition  ProcessEnabledTran NIL
      (* check for source, transitions with both input and output places as well as sinks)]

[METH Transition  RemoveSelf NIL
      (* Will remove itself and any connecting arcs)]

[METH VTransition  AddInArc (ArcInst)
      (* Add an incoming arc into VTrans)]

[METH VTransition  AddOutArc (ArcInst)
      (* Add an outgoing arc from VTrans)]
```

```
[METH VTransition  AddToInputFn (Place)
      (* Add place instance into input function for VTrans)]

[METH VTransition  AddToOutputFn (Place)
      (* Add place instance into output function for VTrans)]

[METH VTransition  Draw (Center)
      (* Draw a vertical transition and update the global transition list)]

(DEFINEQ

(HTransition.AddInArc
  [Method ((HTransition AddInArc)
           self ArcInst)                                    (* edited "30-Jun-87 08:21")
                                                            (* add new arc drawn into InArcs instance)
          (_@
            :InArcs
            (CONS ArcInst (@ :InArcs])

(HTransition.AddOutArc
  [Method ((HTransition AddOutArc)
           self ArcInst)                                    (* edited "30-Jun-87 08:21")
                                                            (* adds new arc drawn into OutArcs instance)
          (_@
            :OutArcs
            (CONS ArcInst (@ :OutArcs])

(HTransition.AddToInputFn
  [Method ((HTransition AddToInputFn)
           self Placa)                                      (* adited: "13-Aug-87 09:53")
                                                            (* Adds horizontal transition into the list of Place)
          (_@
            :InputList
            (CONS Place (@ :InputList])

(HTransition.AddToOutputFn
  [Method ((HTransition AddToOutputFn)
           self Place)                                      (* edited: "13-Aug-87 09:54")
                                                            (* Adds horizontal transition into output list of
                                                             Place)
          (_@
            OutputList
            (CONS Place (@ :OutputList])

(HTransition.Draw
  [Method ((HTransition Draw)
           self Center)                                     (* adited: "25-Mar-88 14:13")
          (_@
            :Centar Centar)
          (_@
            :Region
            (LIST (DIFFERENCE (CAR (@ :Cantar))
                              (@ ::HalfRagionLength))
                  (OIFFERENCE (CDR (@ :Canter))
                              (@ ::Width))
                  (@ ::RagionLangth)
                  (@ ::RegionWidth)))
```

```
(DSK)<LISPFILES>TRANSFILE.;1  (HTransition.Draw cont.)                                  Page    4

          (_@
            :FlashRegion
            (LIST (DIFFERENCE (CAR (@ :Center))
                              (@ ::HalfRegionLength))
                  (DIFFERENCE (CDR (@ :Center))
                              (@ ::Width))
                  (@ ::RegionLength)
                  (@ ::RegionWidth)))
          (LET ((Label NIL))
               (COND
                 ((NOT (_ self DoesRegionOverlap (@ :Region)))
                   (DRAWLINE (PLUS (CAR (@ :Center))
                                   (@ ::Length))
                             (CDR (@ :Center))
                             (DIFFERENCE (CAR (@ :Center))
                                         (@ ::Length))
                             (CDR (@ :Center))
                             (@ ::Thickness)
                             NIL
                             (@ ::MainW))
                   (_@
                     ::TransCount
                     (ADD1 (@ ::TransCount)))
                   [if (NULL (@ ::TransDeleteFlag))
                       then (SETQ Label (PACK* (QUOTE t)
                                               (@ ::TransCount)))
                     else (SETQ Label (PACK* (QUOTE t)
                                             (_ self FindCorrectLabel]
                   (MOVETO (PLUS (CAR (@ :Center))
                                 22)
                           (DIFFERENCE (CDR (@ :Center))
                                       3)
                           (@ ::MainW))
                   (PRIN1 Label (@ ::MainW))
                   (_@
                     :LabelRegion
                     (LIST (PLUS (CAR (@ :Center))
                                 18)
                           (DIFFERENCE (CDR (@ :Center))
                                       5)
                           24 10))
                   (_@
                     ::SizeFlag T)
                   (_@
                     ::TransLabels
                     (CONS Label (@ ::TransLabels)))
                   (_@
                     :TransName Label)
                   (_@
                     ::TransList
                     (CONS self (@ ::TransList)))
                   (CLEARW (@ ::PromptW)))
                 (T (PRINTBELLS)
                    (_ self PutPrompt "Put the Object somewhere else"])

(Transition.CheckForEnable
  (Method ((Transition CheckForEnable)
           self)                                         (* edited: "23-Sep-87 19:12")
                                                         (* check each place coming into a transition to see if
                                                          the transition is enabled)
          (LET ((InList (@ :InputList))
                (OutList (@ :OutputList))
                (Enabled NIL)
                (Place NIL)
```

```
                (Fine T)
                (Stop NIL))
               (while (AND (NOT (NULL OutList))
                               (NOT Stop))
                  do (SETQ Place (CAR OutList))
                       (SETQ Fine (_ Place CheckForBound))
                       (SETQ OutList (COR OutList))
                       (if (NOT Fine)
                           then (SETQ Stop T)
                                  (_ self PutPrompt "Bound reached for this place")))
               (if Fine
                   then (SETQ Stop NIL)
                          (while (AND (NOT (NULL InList))
                                          (NOT Stop))
                             do (SETQ Place (CAR InList))
                                  (SETQ Enabled (_ Place CheckForToken))
                                  (if Enabled
                                      then (SETQ Enabled NIL)
                                             (SETQ InList (CDR InList))
                                    else (SETQ Stop T)]
               (if Stop
                   then NIL
                 else self))))

(Transition.CheckInputPlaces
  [Method ((Transition CheckInputPlaces)
            self)                                        (* edited: " 9-Mar-88 20:41 ")
                                                         (* New method template)
          (LET ((InputPlaces (@ :InputList))
                (Place NIL))
               (while (NOT (NULL InputPlaces))
                  do (SETQ Place (CAR InputPlaces))
                       (if (EQ (@ Place :Tokens)
                                 0)
                           than (_ Place DrawANewToken))
                       (SETQ InputPlacas (CDR InputPlaces])

(Transition.ChackNumArcs
  (Method ((Transition CheckNumArcs)
            self)                                        (* edited: " 5-Jan-88 11:31")
                                                         (* New method template)
          (LET ((Num 0)
                (List (@ :InArcs)))
               (whila (NOT (NULL List))
                  do (SETQ List (COR List))
                       (SETQ Num (PLUS Num 1)))
            Num)))

(Transition.DalateFromInputList
  [Method ((Transition DeleteFromInputList)
            self Place)                                  (* edited: "30-Mar-88 08:38")
                                                         (* remove tha Place instance from input list of
                                                          transition)
          [LET ((Obj NIL)
                (List (@ :InputList)))
               (_@
                 :InputList NIL)
               (whila (NOT (NULL List)) do ((SETQ Obj (CAR List))
                                             [If (NEQ Obj Placa)
                                                 then (_@
                                                        :InputList
                                                        (CONS Obj (@ :InputList]
                                             (SETQ List (COR List]
          (If (AND (NULL (@ :InputList))
```

```
                    (NULL (@ :OutputList)))
            then (if (EQ (@ :Type)
                         (QUOTE VTrans))
                     then (DRAWLINE (CAR (@ :Center))
                                    (PLUS (CDR (@ :Center))
                                          (@ ::Length))
                                    (CAR (@ :Center))
                                    (DIFFERENCE (CDR (@ :Center))
                                                (@ ::Length))
                                    (@ ::Width)
                                    (QUOTE ERASE)
                                    (@ ::MainW))
                   else (DRAWLINE (PLUS (CAR (@ :Center))
                                        (@ ::Length))
                                  (CDR (@ :Center))
                                  (DIFFERENCE (CAR (@ :Center))
                                              (@ ::Length))
                                  (CDR (@ :Center))
                                  (@ ::Width)
                                  (QUOTE ERASE)
                                  (@ ::MainW)))
                 (_ self RemoveSelf)
          else (if (EQ (@ :Type)
                       (QUOTE VTrans))
                   then (DRAWLINE (CAR (@ :Center))
                                  (PLUS (CDR (@ :Center))
                                        (@ ::Length))
                                  (CAR (@ :Center))
                                  (DIFFERENCE (CDR (@ :Center))
                                              (@ ::Length))
                                  (@ ::Width)
                                  NIL
                                  (@ ::MainW))
                 else (DRAWLINE (PLUS (CAR (@ :Center))
                                      (@ ::Length))
                                (CDR (@ :Center))
                                (DIFFERENCE (CAR (@ :Center))
                                            (@ ::Length))
                                (CDR (@ :Center))
                                (@ ::Width)
                                NIL
                                (@ ::MainW])

(Transition.DeleteFromOutputList
  [Method ((Transition DeleteFromOutputList)
           self Place)                                 (* edited: "30-Mar-88 08:38")
                                                       (* remove the Place instance from output list of
                                                        transition)
         [LET ((Obj NIL)
               (List (@ :OutputList)))
              (_@
                :OutputList NIL)
              (while (NOT (NULL List)) do ((SETQ Obj (CAR List))
                                           [if (NEQ Obj Place)
                                               then (_@
                                                      :OutputList
                                                      (CONS Obj (@ :OutputList]
                                           (SETQ List (CDR List]
         (if (AND (NULL (@ :InputList))
                  (NULL (@ :OutputList)))
             then (if (EQ (@ :Type)
                          (QUOTE VTrans))
                      then (DRAWLINE (CAR (@ :Center))
                                     (PLUS (CDR (@ :Center))
```

```
                                                        (@ ::Length))
                                                   (CAR (@ :Center))
                                                   (DIFFERENCE (CDR (@ :Center))
                                                               (@ ::Length))
                                                   (@ ::Width)
                                                   (QUOTE ERASE)
                                                   (@ ::MainW))
                             else (DRAWLINE (PLUS (CAR (@ :Center))
                                                      (@ ::Length))
                                                 (CDR (@ :Center))
                                                 (DIFFERENCE (CAR (@ :Center))
                                                             (@ ::Length))
                                                 (CDR (@ :Center))
                                                 (@ ::Width)
                                                 (QUOTE ERASE)
                                                 (@ ::MainW)))
                            (_ self RemoveSelf)
                  else (if (EQ (@ :Type)
                                  (QUOTE VTrans))
                             then (DRAWLINE (CAR (@ :Center))
                                                 (PLUS (CDR (@ :Center))
                                                       (@ ::Length))
                                                 (CAR (@ :Center))
                                                 (DIFFERENCE (CDR (@ :Center))
                                                             (@ ::Length))
                                                 (@ ::Width)
                                                 NIL
                                                 (@ ::MainW))
                           else (DRAWLINE (PLUS (CAR (@ :Center))
                                                    (@ ::Length))
                                               (CDR (@ :Center))
                                               (DIFFERENCE (CAR (@ :Center))
                                                           (@ ::Length))
                                               (CDR (@ :Center))
                                               (@ ::Width)
                                               NIL
                                               (@ ::MainW])

(Transition.DoUpdate
  [Method ((Transition DoUpdate)
           self size)                                    (* edited: "22-Mar-88 20:22")
                                                         (* New method template)
          (COND
            ((EQ size (QUOTE Small))
              (_@
                ::Length 13)
              (_@
                ::Width 2)
              (_@
                ::RegionLength 30)
              (_@
                ::RegionWidth 4)
              (_@
                ::HalfRegionLength 15)
              (_@
                ::Thickness 2))
            ((EQ size (QUOTE Medium))
              (_@
                ::Length 16)
              (_@
                ::Width 3)
              (_@
                ::RegionLength 36)
              (_@
```

```
                  ::RegionWidth 6)
                (_@
                  ::HalfRegionLength 18)
                (_@
                  ::Thickness 3))
              ((EQ size (QUOTE Big))
                (_@
                  ::Length 19)
                (_@
                  ::Width 4)
                (_@
                  ::RegionLength 42)
                (_@
                  ::RegionWidth 8)
                (_@
                  ::HalfRegionLength 22)
                (_@
                  ::Thickness 4])

(Transition.FindCorrectLabel
  (Method ((Transition FindCorrectLabel)
           self)                                              (* edited: "22-Mar-88 20:11")
                                                              (* New method template)
         (LET ((List (@ ::TransList))
               (Done NIL)
               (Label NIL)
               (Count 1)
               (Found NIL)
               (CurTrans NIL))
              (SETQ CurTrans (CAR List))
              (while (NOT Found)
                 do (SETQ Done NIL)
                    [while (AND (NOT Done)
                                (NOT (NULL List)))
                       do (SETQ List (CDR List))
                          (SETQ Label (UNPACK (@ CurTrans :TransName)))
                          (if (NOT (NULL (CDDR Label)))
                              then (SETQ Number (PACK* (CADR Label)
                                                       (CADDR Label)))
                            else (SETQ Number (CADR Label)))
                          (if (EQP Count Number)
                              then (SETQ Done T)
                                   (SETQ Count (ADD1 Count))
                                   (SETQ List (@ ::TransList))
                                   (SETQ CurTrans (CAR List))
                            else (SETQ CurTrans (CAR List]
                    (if (AND (NULL List)
                             (NOT Done))
                        then (SETQ Found T)))
          Count)))

(Transition.HighLightSelf
  [Method ((Transition HighLightSelf)
           self)                                              (* edited: "24-Nov-87 10:41")
                                                              (* New method tamplate)
         (SHADEGRIDBOX 0 0 0 NIL (@ :FlashRegion)
                       NIL
                       (@ ::MainW))
         (BLOCK 100)
         (if (EQ (@ Type)
                 (QUOTE VTrans))
             then (DRAWLINE (CAR (@ :Center))
                            (PLUS (CDR (@ :Center))
                                  (@ ::Length))
```

```
                                          (CAR (@ :Center))
                                          (DIFFERENCE (COR (@ :Center))
                                                      (@ ::Length))
                                          (@ ::Thickness)
                                          NIL
                                          (@ ::MainW))
            else (DRAWLINE (PLUS (CAR (@ :Center))
                                 (@ ::Length))
                                 (COR (@ :Center))
                                 (DIFFERENCE (CAR (@ :Center))
                                             (@ ::Length))
                                 (COR (@ :Center))
                                 (@ ::Thickness)
                                 NIL
                                 (@ ::MainW])

(Transition.ProcessEnabledTran
  [Method ((Transition ProcessEnabledTran)
           self)                                         (* edited: "11-Feb-88 13:38")
                                                         (* check for source, transitions with both input and
                                                          output places as well as sinks)
         (LET ((CurPlace NIL)
               (InputPlaces (@ :InputList))
               (OutputPlaces (@ :OutputList)))
              (if (AND (NOT (NULL InputPlaces))
                       (NOT (NULL OutputPlaces)))
                  then (_ self HighLightSelf)
                       (BLOCK 200)
                       (_ self HighLightSelf)
                       (while (NOT (NULL InputPlaces))
                          do (SETQ CurPlace (CAR InputPlaces))
                             (_ CurPlace DeleteAToken)
                             (SETQ InputPlaces (COR InputPlaces)))
                       (while (NOT (NULL OutputPlaces))
                          do (SETQ CurPlace (CAR OutputPlaces))
                             (_ CurPlace DrawANewToken)
                             (SETQ OutputPlaces (COR OutputPlaces)))
                else (if (NOT (NULL OutputPlaces))
                         then (while (NOT (NULL OutputPlaces))
                                 do (_ self HighLightSelf)
                                    (BLOCK 200)
                                    (_ self HighLightSelf)
                                    (SETQ CurPlace (CAR OutputPlaces))
                                    (_ CurPlace DrawANewToken)
                                    (SETQ OutputPlaces (COR OutputPlaces))
                                    (_ self PutPrompt "Source provided a token"))
                       else (if (NULL OutputPlaces)
                                then (_ self HighLightSelf)
                                     (BLOCK 200)
                                     (_ self HighLightSelf)
                                     (while (NOT (NULL InputPlaces))
                                        do (SETQ CurPlace (CAR InputPlaces))
                                           (_ CurPlace DeleteAToken)
                                           (SETQ InputPlaces (CDR InputPlaces)))
                                     (_ self PutPrompt "token went into the sink"])

(Transition.RemoveSelf
  [Method ((Transition RemoveSelf)
           self)                                         (* edited: "25-Mar-88 15:08")
                                                         (* Will remove itself and any connecting arcs)
         (_@
           ::TransDeleteFlag T)
         (SHADEGRIDBOX D O D NIL (@ :Region)
                       NIL
```

```
                    (@ ::MainW))
          (SHADEGRIDBOX 0 0 0 NIL (@ :LabelRegion)
                        NIL
                        (@ ::MainW))
          [LET ((Obj NIL)
                (List (@ ::TransList)))
               (_@
                 ::TransList NIL)
               (while (NOT (NULL List)) do ((SETQ Obj (CAR List))
                                              [if (NEQ Obj self)
                                                  then (_@
                                                         ::TransList
                                                         (CONS Obj (@ ::TransList]
                                              (SETQ List (CDR List]
          [if (NOT (NULL (@ :InArcs)))
              then (LET ((List (@ :InArcs))
                         (CurArc NIL))
                        (while (NOT (NULL List))
                           do (SETQ CurArc (CAR List))
                              (SETQ List (CDR List))
                              (_ CurArc RemoveSelf]
          (_@
            :InArcs NIL)
          [if (NOT (NULL (@ :OutArcs)))
              then (LET ((List (@ :OutArcs))
                         (CurArc NIL))
                        (while (NOT (NULL List))
                           do (SETQ CurArc (CAR List))
                              (SETQ List (CDR List))
                              (_ CurArc RemoveSelf]
          (_@
            :OutArcs NIL)
          [if (NOT (NULL (@ :InputList)))
              then (LET ((List (@ :InputList))
                         (Obj NIL))
                        (while (NOT (NULL List))
                           do (SETQ Obj (CAR List))
                              (SETQ List (CDR List))
                              (_ Obj DeleteFromOutputList self]
          (_@
            :InputList NIL)
          [if (NOT (NULL (@ :OutputList)))
              then (LET ((List (@ :OutputList))
                         (Obj NIL))
                        (while (NOT (NULL List))
                           do (SETQ Obj (CAR List))
                              (SETQ List (CDR List))
                              (_ Obj DeleteFromInputList self]
          (_@
            :OutputList NIL)
          [LET ((Label (@ :TransName))
                (List (@ ::TransLabels)))
               (_@
                 ::TransLabels NIL)
               (while (NOT (NULL List)) do ([if (NEQ Label (CAR List))
                                                  then (_@
                                                         ::TransLabels
                                                         (CONS (CAR List)
                                                               (@ ::TransLabels]
                                              (SETQ List (CDR List]
          (_@
            ::TransCount
            (SUB1 (@ ::TransCount)))
          (_@
```

```
              :Region
              (QUOTE (0 0 0 0)))
            (_@
              :Center
              (QUOTE (0.0])

(VTransition.AddInArc
  [Method ((VTransition AddInArc)
            self ArcInst)                                      (* edited: "15-Jul-87 08:06")
                                                               (* Add an incoming arc into VTrans)
            (_@
              :InArcs
              (CONS ArcInst (@ :InArcs])

(VTransition.AddOutArc
  [Method ((VTransition AddOutArc)
            self ArcInst)                                      (* edited: "15-Jul-87 08:09")
                                                               (* Add an outgoing arc from VTrans)
            (_@
              :OutArcs
              (CONS ArcInst (@ :OutArcs])

(VTransition.AddToInputFn
  [Method ((VTransition AddToInputFn)
            self Place)                                        (* edited: "15-Jul-87 08:08")
                                                               (* Add place instance into input function for VTrans)
            (_@
              :InputList
              (CONS Place (@ :InputList])

(VTransition.AddToOutputFn
  [Method ((VTransition AddToOutputFn)
            self Place)                                        (* edited: "15-Jul-87 08:08")
                                                               (* Add place instance into output function for VTrans)
            (_@
              :OutputList
              (CONS Place (@ :OutputList])

(VTransition.Draw
  [Method ((VTransition Draw)
            self Center)                                       (* edited: "25-Mar-88 14:13")
            (_@
              :Center Center)
            (_@
              :Region
              (LIST (DIFFERENCE (CAR (@ :Center))
                                (@ ::Width))
                    (DIFFERENCE (CDR (@ :Center))
                                (@ ::HalfRegionLength))
                    (@ ::RegionWidth)
                    (@ ::RegionLength)))
            (_@
              :FlashRegion
              (@ :Region))
            (LET ((Label NIL))
                 (COND
                   ((NOT (_ self DoesRegionOverlap (@ :Region)))
                     (DRAWLINE (CAR (@ :Center))
                               (PLUS (CDR (@ :Center))
                                     (@ ::Length))
                               (CAR (@ :Center))
                               (DIFFERENCE (CDR (@ :Center))
                                           (@ ::Length))
                               (@ ::Thickness)
```

```
                                 NIL
                                 (@ ::MainW))
                        (_@
                          ::TransCount
                          (ADD1 (@ ::TransCount)))
                        [if (NULL (@ ::TransDeleteFlag))
                            then (SETQ Label (PACK* (QUOTE t)
                                                    (@ ::TransCount)))
                          else (SETQ Label (PACK* (QUOTE t)
                                                  (_ self FindCorrectLabel]
                        (MOVETO (DIFFERENCE (CAR (@ :Center))
                                            5)
                                (DIFFERENCE (CDR (@ :Center))
                                            29)
                                (@ ::MainW))
                        (PRIN1 Label (@ ::MainW))
                        (_@
                          :LabelRegion
                          (LIST (DIFFERENCE (CAR (@ :Center))
                                            4)
                                (DIFFERENCE (CDR (@ :Center))
                                            30)
                                22 9))
                        (_@
                          ::SizeFlag T)
                        (_@
                          ::TransLabels
                          (CONS Label (@ ::TransLabels)))
                        (_@
                          :TransName Label)
                        (_@
                          ::TransList
                          (CONS self (@ ::TransList)))
                        (CLEARW (@ ::Promptw)))
                      (T (PRINTBELLS)
                         (_ self PutPrompt "Put the Object somewhere else"]))
(PUTPROPS TRANSFILE COPYRIGHT ("Xerox Corporation" 1988))
(DECLARE: DONTCOPY
  (FILEMAP (NIL (5059 24850 (HTransition.AddInArc 5069 . 5380) (HTransition.AddOutArc 5382 . 5699) (
HTransition.AddToInputFn 5701 . 6035) (HTransition.AddToOutputFn 6037 . 6394) (HTransition.Draw 6396
 . 8332) (Transition.CheckForEnable 8334 . 9562) (Transition.CheckInputPlaces 9564 . 10120) (
Transition.CheckNumArcs 10122 . 10543) (Transition.DeleteFromInputList 10545 . 12488) (
Transition.DeleteFromOutputList 12490 . 14440) (Transition.DoUpdate 14442 . 15333) (
Transition.FindCorrectLabel 15335 . 16543) (Transition.HighLightSelf 16545 . 17476) (
Transition.ProcessEnabledTran 17478 . 19329) (Transition.RemoveSelf 19331 . 21787) (
VTransition.AddInArc 21789 . 22093) (VTransition.AddOutArc 22095 . 22403) (VTransition.AddToInputFn
22405 . 22739) (VTransition.AddToOutputFn 22741 . 23080) (VTransition.Draw 23082 . 24848)))))
STOP
```

{OSK}<LISPFILES>TRANSFILE.;1

{DSK}<LISPFILES>TRANSFILE.;1 6-Apr-88 09:06:15
-- Listed on 6-Apr-88 09:08:28 --

FUNCTION INDEX

A GRAPHICAL ENVIRONMENT
FOR THE SIMULATION OF PETRI NETS

by

TAN, JOO TONG

B.S., University of New Mexico, 1986

AN ABSTRACT OF A THESIS

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computing and Information Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1988

ABSTRACT

Real systems exhibit an enormous amount of complexity unless examined at an abstract level. As a result, macroscopic abstractions are needed in order to master this complexity and to better understand a system. Petri nets have developed over the last decade into a suitable model for modeling concurrent systems. Petri nets have a firm theoretical basis on which nets on a higher level are based on the solid foundation provided by lower level nets. A major obstacle to the use of diagrams in modeling systems is often the time and effort required to draw and edit such diagrams. Moreover, it is not possible to see the actual simulation of a system without the aid of a computerized tool. Therefore, the graphical simulation of Petri nets is often more suitable in illustrating the concepts of modeling systems. Hence, computer tools should be available to help users with the modeling and simulation of Petri nets. We developed a Petri net tool package to facilitate the drawing, simulation, and analysis of Petri nets. Our tool package named PETRISYS, consists of a graphical editor, a net simulator, and an analyzer. The graphical editor enables users to edit Petri net models of systems and make changes to them easily. The net simulator runs the system being modeled in 'real-time'. Users can watch the execution of a Petri net as it runs on the screen. The movement of tokens are actually shown on the screen. The net analyzer provides certain properties of the modeled system by making use of a reachability tree which is created internally. The properties that are analyzed from this tree are Safeness, Boundedness, and Conservation. PETRISYS is implemented in an Object-Oriented manner using the programming language LOOPS to ease the process of development. LOOPS runs on top of the Interlisp-D environment and is available on the Xerox 1186 Artificial Intelligence workstation. PETRISYS constitutes the basic features of a possibly powerful and sophisticated Petri net tool package. Many extensions can be added to PETRISYS and these extensions should be made to utilize the full capabilities of a well-designed system.